Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
“Санкт-Петербургский государственный университет”
Математическое обеспечение и администрирование информационных систем
Кафедра информационно-аналитических систем

Шквиро Ирина Алексеевна

Оценка применимости HMM для анализа потоков данных

Выпускная квалификационная работа

Научный руководитель:
Доктор физико-математических наук,
Профессор Новиков Б. А.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics and Mechanics Faculty
SubDepartment of Analytical Information Systems

Shkviro Irina

Evaluation of applicability of HMM for analysis of data streams

Graduate qualification work

Scientific advisor
Professor Boris Novikov

Saint-Petersburg
2016

Оглавление

Введение 6
Мотивация 6
Постановка задачи 6
Обзор существующих решений 8
Поиск оптимального решения. Метод градиентного спуска 8
Стандартный градиентный спуск 8
Стохастический градиентный спуск 8
Метод сопряженных градиентов 9
Многослойный персептрон 11
Автоэнкодеры 12
Стандартный автоэнкодер 12
Шумоподавляющий автоэнкодер 13
Стековый шумоподавляющий автоэнкодер 14
Классификаторы 15
Логистическая регрессия 15
HMM 16
Часть 1. Задача распознавания активностей 22
Использование описанных сетей 22
Подходы к использованию HMM 22
Распознавание отдельных слов (word recognition) 22
Распознавание речи (speech recognition) 23
Использование HMM в данной работе 23
Связь автоэнкодера и классификатора 24
Реализация 24

Эксперименты 25
Исходные данные 25
Подбор параметров стекового шумоподавляющего автоэнкодера 25
Сравнение стохастического градиентного спуска и метода сопряженных градиентов……..25
Анализ влияния размера окна на результат……28
Логистическая регрессия 32
Модель классификатора……....32
Модель стекового шумоподавляющего автоэнкодера с классификатором на вернем слое……34
HMM 35
Подготовка данных……...35
Подбор параметров и результаты экспериментов……..36
Выводы по задаче распознавания активностей 42
Часть 2. Задача распознавания речи 44
Постановка задачи 44
Модель нейросети и методы ее обучения 44
Подходы к обучению нейросетей 44
Стандартный подход…….44
Соревновательный подход……...45
Гибрид……46
Модель нейронной сети 47
Реализация 49
Эксперименты 49
Исходные данные 49

Сравнение подходов 49
Стадия pre-training……50
Стадия fine-tuning……..52
Обучение модели после стандартного pre-training……52
Обучение модели после соревновательного pre-training……...53
Стандартный подход на стадии fine-tuning……54
Соревновательный подход на стадии fine-tuning……...55
Сравнение размеров mini-batches……55
Выводы по задаче распознавания речи 55
Заключение 57
Список литературы 58

Введение

Мотивация

В задачах выявления некоторых зависимостей между входными и выходными данными хорошо себя зарекомендовали нейронные сети. Чтобы достичь наилучших результатов, нужно найти подходящий для задачи тип сети и обучить ее наиболее оптимальным способом.

Для выявления различных закономерностей в данных используется множество различных сетей. Области применения некоторых из них изучены довольно обширно, области применения других – повод для новых исследований. Актуальны задачи анализа потоковых данных. К такому типу задач относятся: распознавание речи, распознавание слов, распознавание типа активности, определение фаз сна и другие задачи. В частности, для решения задач распознавания речи и отдельных слов успешно применяются Скрытые Марковские Модели (Hidden Markov Model, HMM). В связи с этим возникает вопрос о применимости данного типа сетей к другим задачам анализа потоковых данных.

Данная работа направлена на изучение HMM, анализа применимости данного типа сетей к задаче анализа потока данных, а также выборе наиболее оптимального способа ее обучения. Поставлена задача распознавания фаз сна человека по имеющемуся размеченному набору данных. Ожидается, что после обучения нейронная сеть сможет определять фазы сна по данным, представленным в таком же формате.

Выбор наиболее оптимального способа обучения сети, в данном контексте может быть сформулирован следующим образом: “Как наиболее быстро найти параметры модели, при которых нейросеть будет решать поставленную задачу с необходимой точностью?”.

Постановка задачи

Имеется размеченный набор данных об активности человека (замеры показаний сенсоров в последовательные моменты времени и метки,

характеризующие фазы сна). Необходимо создать модель классификатора, которая по аналогичным данным сможет распознавать фазы сна человека в каждый момент времени. В качестве классификатора в модели (одной или нескольких) должна быть использована HMM.

Также нужно рассмотреть различные способы обучения нейронной сети, сравнить их с точки зрения качества и скорости обучения (на той же или другой модели и входных данных).

Обзор существующих решений

В этом разделе мы рассмотрим устройство некоторых нейронных сетей, которые могут быть использованы в нашей задаче, а также основные понятия, связанные с нейросетями.

Поиск оптимального решения. Метод градиентного спуска

Стандартный градиентный спуск

Обычный градиентный спуск – алгоритм, который итеративно спускается по поверхности ошибки к точке локального минимума. Другими словами, это алгоритм минимизации функции в направлении градиента.

Таким образом, задача состоит в минимизации некоторой целевой функции

$$Q(\theta) = \sum_{i=1}^{n} Q_i(\theta)$$

На каждом шаге обновляется значение θ по следующей формуле:

$$\theta = \theta - \alpha \sum_{i=1}^{n} \frac{\partial Q_i(\theta)}{\partial \theta}$$

где параметр α – скорость обучения.

Стохастический градиентный спуск

Стохастический градиентный спуск основывается на тех же принципах, но результат достигается быстрее, так как оценка производится всего лишь на нескольких примерах вместо всего тренировочного множества.

Имеет место и другой вариант – “mini-batches”, который является логическим продолжением стохастического градиентного спуска. Он аналогичен стохастическому градиентному спуску, но в этом методе для оценки градиента вместо одного тренировочного примера используется некоторое подмножество примеров (mini-batch).

Метод сопряженных градиентов

Метод сопряженных градиентов – один из наиболее популярных и хорошо известных итеративных методов для решения систем линейных алгебраических уравнений. Задача состоит в минимизации тестовой функции:

$$\varphi(x) = \frac{1}{2}x^T Ax - x^T b$$

где $b, x \epsilon R^n$, $A \epsilon R^{n \times n}$ и A – симметричная положительно определенная матрица.

Пусть x^* - точка минимума функции φ, тогда

$$\nabla\varphi(x^*) = Ax^* - b = 0$$

или

$$Ax^* = b$$

То есть точка минимума функции $\varphi(x)$ также является решением системы линейных уравнений $Ax = b$, что позволяет минимизировать функцию $\varphi(x)$ вместо прямого поиска решения системы уравнений. Единственность решения гарантируется ограничениями, наложенными на матрицу *A*.

Идея метода состоит в следующем: выбираем начальную позицию x_0, затем на каждом шаге движемся в направлении, таком, что $\varphi(x_{k+1}) < \varphi(x_k)$. Причем $x_{k+1} = x_k + \alpha_k p_k$, где α_k – длина шага, p_k – направление поиска; выбор α_k и p_k зависит от конкретного метода.

Введем обозначение

$$r_k = \nabla\varphi(x_k) = Ax_k - b$$

Длина шага выбирается следующим образом:

$$\alpha_k = \frac{\nabla\varphi(x_k)^T \nabla\varphi(x_k)}{\nabla\varphi(x_k)^T A \nabla\varphi(x_k)} = \frac{{r_k}^T r_k}{{r_k}^T A r_k}$$

Будем искать теперь n линейно независимых векторов $\{p_0, p_1, \dots, p_{n-1}\}$ – базис R^n. Тогда можно минимизировать функцию $\varphi(x)$, минимизируя ее вдоль каждой оси. Тогда, так как $\{p_0, p_1, \dots, p_{n-1}\}$ – базис R^n, разность между

точным решением x^* и первым приближением x_0 может быть выражена следующим образом:

$$x^* - x_0 = \sigma_0 p_0 + \sigma_1 p_1 + \cdots + \sigma_{n-1} p_{n-1}$$

Заметим, что σ_k совпадают с α_k, тогда

$$x^* = x_0 + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_{n-1} p_{n-1}$$

Для диагональных матриц *A* на каждом шаге x_k – проекция точного решения x^* на пространство, покрытое *k* векторами.

Другое полезное свойство: каждый следующий сопряженный вектор p_k может быть получен, опираясь только на предыдущий вектор p_{k-1}, без знания о предыдущих векторах. Новый вектор автоматически будет ортогонален каждому из них. Формула для направления на *k*-ом шаге:

$$p_k = -r_k + \beta_k p_{k-1}$$

где

$$\beta_k = \frac{{r_k}^T A p_{k-1}}{{p_{k-1}}^T A p_{k-1}} = \frac{{r_k}^T r_k}{{r_{k-1}}^T r_{k-1}}$$

Весь алгоритм выглядит следующим образом:

1. Подготовка:
    - Выбор начального приближения x_0
    - $r_0 = A x_0 - b$
    - $p_0 = -r_0$
2. (*k+1*)-ая итерация:
    - $\alpha_{k+1} = \frac{{r_k}^T r_k}{{p_k}^T A p_k}$
    - $x_{k+1} = x_k + \alpha_{k+1} p_k$
    - $r_{k+1} = r_k + \alpha_{k+1} A p_k$
    - $\beta_{k+1} = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$
    - $p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$

Многослойный персептрон

Формальные нейроны могут соединяться в сети различными способами. Самый распространенный – многослойный персептрон (нейроны каждого слоя соединяются с нейронами предыдущего и следующего слоев по принципу “каждый с каждым”). Многослойный персептрон рассчитывает выходной вектор *y* для любого входного вектора *x*, то есть формирует отображение $\varphi: X \to Y \; \forall x \in X$. С его помощью решаются такие распространенные задачи, как задача классификации, распознавания, прогнозирования одномерной функции, аппроксимация многомерной функции. Для каждой задачи важно выбрать оптимальное количество нейронов и слоев. Выбор будет обусловлен сложностью задачи, количеством данных для обучения, требуемым количеством входов и выходов сети, а также техническими возможностями машины, на которой моделируется сеть. Стоит уделить особое внимание подготовке входных и выходных данных с помощью масштабирования или нелинейных преобразований.

Далее, для облегчения понимания, мы будем рассматривать модель многослойного персептрона с одним скрытым слоем.

Многослойный персептрон можно рассматривать как функцию $f: R^D \longrightarrow R^L$, где *D* – размерность входного вектора *x*, *L* – размерность выходного вектора *f*(*x*). Тогда *f*(*x*) можно определить в терминах матриц:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x)))$$

где $b^{(1)}, b^{(2)}$ – вектора смещения, $W^{(1)}, W^{(2)}$ – матрицы весов, G, s – функции активации. Вектор $h(x) = \Phi(x) = s(b^{(1)} + W^{(1)}x)$ определяет скрытый уровень. $W^{(1)} \epsilon R^{D \times D_h}$ - весовая матрица, связывающая входной вектор со скрытым слоем. Каждая колонка $W_i^{(1)}$ определяет веса при переходе от входного вектора к i-му узлу. Функцию *s* обычно выбирают гиперболическим тангенсом, которая определяется следующим образом: $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^a}$ или сигмоидой: $\sigma(a) = \frac{1}{1+e^{-a}}$. Обе функции вещественнозначны и определены на

вещественной оси, однако можно естественным образом определить их для векторов, если применять их поэлементно. Выходной вектор получен следующим образом: $o(x) = G(b^{(2)} + W^{(2)}h(x))$.

Таким образом, набор параметров, корректируемых при обучении: $\theta = \{W^{(2)}, b^{(2)}, W^{(1)}, b^{(1)}\}$. При этом можем использовать алгоритм обратного распространения ошибки.

Автоэнкодеры

Стандартный автоэнкодер

Автоэнкодер - алгоритм обучения без учителя, который использует нейронную сеть и метод обратного распространения ошибки для того, чтобы добиться того, что входной вектор признаков вызывал отклик сети, равный входному вектору, т.е. $y = x$ (рис. 1).

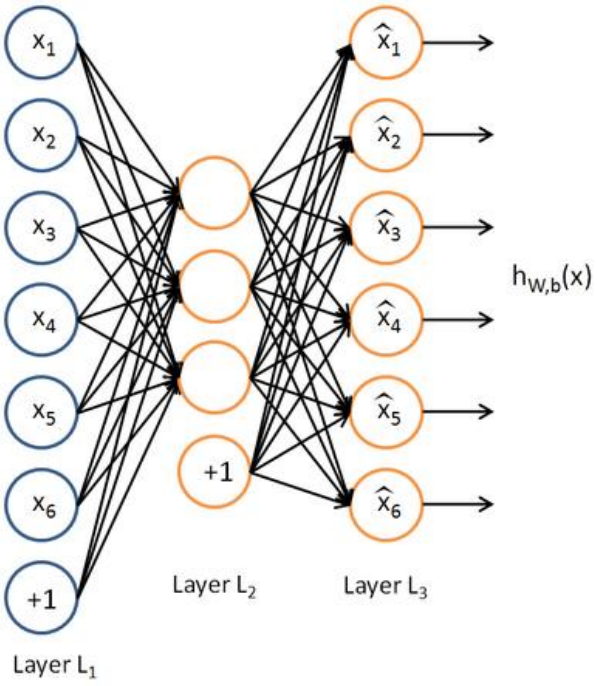


Рис. 1.

Таким образом, задача автоэнкодера сводится к построению функции $y(x)=x$.

Тривиальное решение не удовлетворяет нашей задаче, поэтому необходимо наложить на топологию сети одно из следующих ограничений:

- Количество нейронов скрытого слоя должно быть меньше, чем размерность входных данных (рис. 1).
- Активация нейронов скрытого слоя должна быть разреженной.

Первое ограничение позволяет получить сжатие данных. Такая компрессия возможна, если в данных есть скрытые взаимосвязи, корреляция признаков и т.д. Таким образом, автоэнкодер позволяет понизить размерность входных данных.

Второе ограничение – требование разреженной активации нейронов скрытого слоя, — позволяет получить нетривиальные результаты даже когда количество нейронов скрытого слоя превышает размерность входных данных. Разреженная активация – искусственное ограничение количества одновременно активных нейронов в промежуточном слое.

Значения векторов скрытого слоя вычисляются по формуле: $y = s(Wx + b)$, где s – нелинейная функция. Скрытое представление y затем переводится обратно: $y = s(W'y + b')$, где W', b' – новые параметры, в общем случае, не связанные с W, b. Однако в некоторых случаях используются связанные веса: $W' = W^T$. В качестве целевой функции можно взять следующую:

$$L_H(x, z) = -\sum_{k=1}^{d} [x_k \log z_k + (1 - x_k) \log(1 - z_k)]$$

Понятно, что такая трансформация не может быть одинаково хороша для всех входных векторов. Данная модель дает хорошие результаты на тренировочных образцах и, возможно, на некоторых других, но не на произвольных входных векторах.

Шумоподавляющий автоэнкодер

Основная идея такого автоэнкодера заключается в следующем: чтобы заставить скрытый слой открыть наиболее сильные взаимосвязи, при этом не

строя тождественное отображение, мы учим автоэнкодер восстанавливать вход из его поврежденной версии.

Таким образом, в стандартный автоэнкодер добавляется один шаг, вносящий в исходную последовательность данных шум. На выходе автоэнкодера ожидается получение изначальной (незашумленной) последовательности исходных данных.

Стековый шумоподавляющий автоэнкодер

Стековый автоэнкодер – организация нескольких автоэнкодеров в список, при котором выход автоэнкодера предыдущего уровня подается на вход текущего уровня. Такой вид автоэнкодера, в отличие от обычного шумоподавляющего автоэнкодера имеет две стадии обучения. Первая называется pre-training (предварительное обучение). В течение этой стадии мы обучаем каждый автоэнкодер описанным выше способом. Когда предварительное обучение завершено, начинается вторая стадия обучения, которая называется fine-tuning. Здесь мы тренируем всю сеть, рассматривая ее как единую систему.

Рассмотрим первую стадию обучения (pre-training). Имеется несколько связанных автоэнкодеров. Каждый из них тренируется как обычный автоэнкодер, пытаясь минимизировать ошибку реконструкции входа, который, в свою очередь, является выходом автоэнкодера уровнем ниже. Таким образом, обучение стекового автоэнкодера на первой стадии – последовательное обучение нескольких автоэнкодеров.

Когда мы переходим ко второй стадии (fine-tuning), мы рассматриваем нейросеть как единую систему, то есть на этой стадии система тренируется как обычный многослойный персептрон.

Рассмотрим параметры нейросети, которые мы можем менять для улучшения качества работы сети:

1. Функция активации. Наиболее часто используемые: логистическая функция и гиперболический тангенс.

2. Инициализация весов.
3. Скорость обучения. Простейшее решение – постоянная скорость. Однако, хорошей практикой является уменьшение скорости обучения с течением времени. При использовании данного подхода можно пользоваться формулой: $LearningRate = \frac{\mu_0}{1+dt}$, где μ_0 – начальная скорость, d – некоторая константа, контролирующая уменьшение скорости обучения, t – эпоха.
4. Число скрытых узлов. Этот параметр зависит от данных. Условно говоря, чем "сложнее" данные, тем мощнее сеть нужна для их моделирования и тем большее число узлов скрытого уровня необходимо.
5. Размер окна.
6. Размер mini-batches (в случае обучения "по партиям").

Классификаторы

Логистическая регрессия

Логистическая регрессия – вероятностный линейный классификатор с параметрами W (матрица весов) и b (вектор смещения). Такой классификатор получается путем проектирования входного вектора на множество гиперплоскостей, каждая из которых соответствует некоторому классу.

Запишем вероятность того, что входной вектор x относится к классу i:

$$P(Y = i|x, W, b) = \operatorname*{softmax}_i(Wx + b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j + b_j}}$$

где Y – случайная величина.

В результате модель отнесет исходный вектор к тому классу, для которого такая вероятность максимальна, то есть

$$y_{pred} = \operatorname*{argmax}_i P(Y = i|x, W, b)$$

Для выбора оптимальных параметров модели будем минимизировать функцию ошибки. В случае нескольких классов в качестве функции ошибки

зачастую используется обратная логарифмическая функция правдоподобия. Ее минимизация эквивалентна максимизации функции правдоподобия:

$$L(\theta = \{W, b\}, D) = \sum_{i=0}^{|D|} \log(P(Y = y^i | x^i, W, b))$$

Функция ошибки выглядит следующим образом:

$$l(\theta = \{W, b\}, D) = -L(\theta = \{W, b\}, D)$$

Наиболее простой и часто используемый метод минимизации произвольных нелинейных функций – градиентный спуск, который описан выше.

HMM

Основная задача HMM (Hidden Markov Models, Скрытые Марковские Модели) – распознавание неизвестных процессов на основе наблюдаемых. Значение скрытой переменной $x(t)$ зависит только от значения скрытой переменной $x(t-1)$, значение наблюдаемой переменной $y(t)$ зависит только от значения скрытой переменной $x(t)$ (рис. 2).

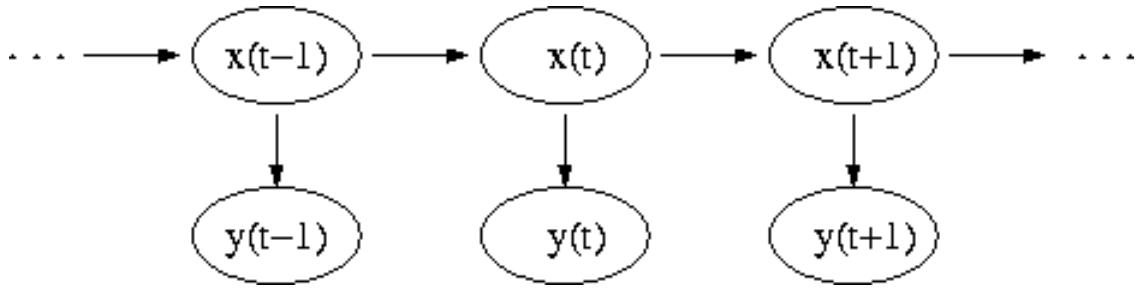


Рис. 2.

Основные элементы модели (рис. 3):

- Наблюдаемые элементы (вектор *Y*)
- Скрытые состояния (вектор *X*)
- Вероятности переходов между состояниями (матрица $A = \{a_{ij}\}$, где a_{ij} – вероятность перехода из состояния i в состояние j)
- Вероятности появления наблюдаемых элементов из каждого состояния (матрица $B = \{b_i(j)\}$, где $b_i(j)$ –вероятность наблюдения элемента j в состоянии i)

- Начальное распределение (вектор, содержащий вероятности появления каждого состояния в начальный момент времени)

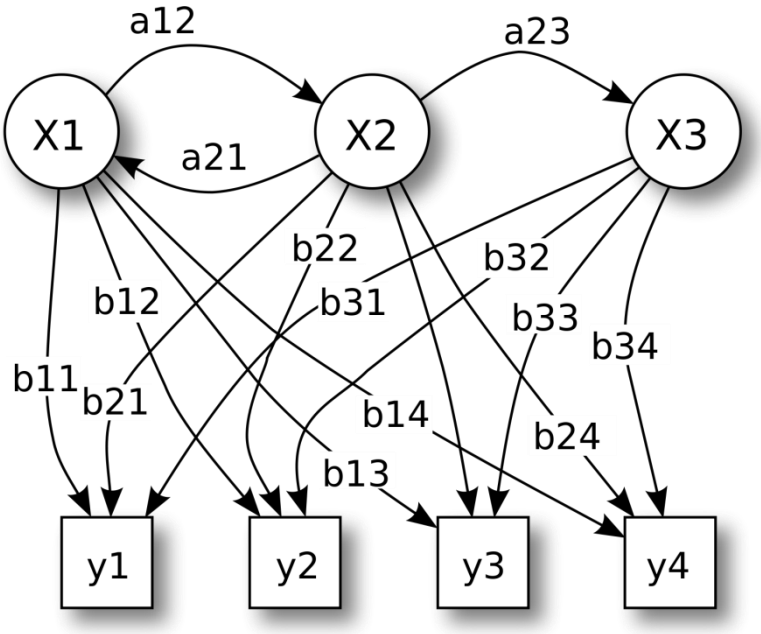


Рис. 3.

Основные вопросы, на которые позволяет ответить HMM:

1. Какова вероятность появления последовательности наблюдений для данной модели?
2. Как выбрать цепочку состояний при заданной модели и наблюдениях, которая наилучшим образом соответствует имеющейся последовательности наблюдений?
3. Каким образом подобрать параметры сети, чтобы вероятность появления данной последовательности наблюдений была наибольшей?

Рассмотрим методы решения каждой из описанных выше задач.

1. Даны модель и последовательность наблюдений. Задача состоит в том, чтобы оценить вероятность появления данной последовательности в данной модели. Можем рассматривать вероятность как оценку качества модели с учетом последовательности (результат может быть использован при выборе из нескольких моделей).

Решение:

Пусть имеется некоторая последовательность наблюдений $O=O_1O_2O_3\ldots O_T$ и модель λ. Обозначим $P(O|\lambda)$ – вероятность появления последовательности O в модели λ. Зафиксируем последовательность состояний $Q=q_1q_2\ldots q_T$, тогда мы можем вычислить вероятность появления наблюдаемой последовательности O для последовательности состояний Q:

$$P(O|Q,\lambda)=\prod_{t=1}^{T}P(O_t|q_t,\lambda)=b_{q_1}(O_1)\cdot b_{q_2}(O_2)\cdot\ldots\cdot b_{q_T}(O_T)$$

$$P(Q|\lambda)=\pi_{q_1}a_{q_1q_2}a_{q_2q_3}\cdot\ldots\cdot a_{q_{T-1}q_T}$$

$$P(O,Q|\lambda)=P(O|Q,\lambda)P(Q|\lambda)$$

$$P(O|\lambda)=\sum_{Q}P(O|Q,\lambda)P(Q|\lambda)$$

$$=\sum_{q_1q_2\ldots q_T}\pi_{q_1}b_{q_1}(O_1)a_{q_1q_2}b_{q_2}(O_2)\cdot\ldots\cdot a_{q_{T-1}q_T}b_{q_T}(O_T)$$

Чтобы посчитать вероятность таким способом потребуется произвести $2TN^T$ вычислений. Рассмотрим алгоритм “прямого-обратного” хода, который позволит сократить количество вычислений.

Алгоритм “прямого-обратного” хода:

Введем $\alpha_t(i)=P(O_1O_2\ldots O_t,q_t=S_i|\lambda)$, то есть $\alpha_t(i)$ – вероятность наблюдения последовательности $O_1O_2\ldots O_t$ и состояния S_i в момент времени t для данной модели λ. Можем определить $\alpha_t(i)$ индуктивно:

Инициализация:

$$\alpha_1(i)=\pi_i b_i(O_1),1\le i\le N$$

Индукция:

$$\alpha_{t+1}(j)=\left[\sum_{i=1}^{N}\alpha_t(i)a_{ij}\right]b_j(O_{t+1}),1\le t\le T-1;\ 1\le j\le N$$

Заключение:

$$P(O|\lambda)=\sum_{i=1}^{N}\alpha_t(i)$$

Такое определение включает порядка N^2T вычислений. Аналогично, можем рассматривать переменную $\beta_t(i) = P(O_{t+1}O_{t+2} \dots O_T | q_t = S_i, \lambda)$ – вероятность части наблюдаемой последовательности с момента времени $t+1$ до конца в состоянии S_i в момент времени t при модели λ. Можем определить $\beta_t(i)$ индуктивно:

Инициализация:

$$\beta_T(i) = 1, 1 \le i \le N$$

Индукция:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(O_{t+1})\beta_{t+1}(j), t = T-1, T-2, \dots, 1;\ 1 \le i \le N$$

Такое определение включает порядка N^2T вычислений.

2. Даны модель и последовательность наблюдений. Задача состоит в том, чтобы найти последовательность скрытых состояний, наилучшим образом описывающих данную последовательность в данной модели. Понятно, что для случаев, кроме вырожденных, “правильной” последовательности нет, поэтому будем использовать некие критерии оптимальности для оценки решений.

Решение:

В отличие от первой задачи, здесь нет точного решения, а основная сложность заключается в выборе критериев оптимальности.

Одно из простейших решений – выбрать состояния q_t, которые по отдельности наиболее вероятны. Для этого введем переменную:

$$\gamma_t(i) = P(q_t = S_i | O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

$$\sum_{i=1}^{N} \gamma_t(i) = 1$$

Тогда наиболее подходящее состояние в момент времени t:

$$q_t = \underset{1 \le i \le N}{\operatorname{argmax}} \gamma_t(i), 1 \le t \le T$$

В такой простой модели может возникнуть проблема: невалидная итоговая последовательность (то есть $a_{ij}=0$ для некоторых i и j). Можно рассмотреть корректные пары состояний ($q_t q_{t+1}$), тройки и т.д. Будем искать наилучший путь методами динамического программирования.

Алгоритм Витерби:

Определим функцию качества: $\delta_t(i) = \max_{q_1 q_2 \dots q_{t-1}} P(q_1 q_2 \dots q_t = i, O_1 O_2 \dots O_t | \lambda)$.

По индукции: $\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] b_j(O_{t+1})$.

Чтобы восстановить последовательность, нужно хранить ряд аргументов $\psi_t(j)$.

Инициализация:

$$\delta_1(i) = \pi_i b_i(O_1), 1 \le i \le N$$

$$\psi_1(i) = 0$$

Индукция:

$$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_t), 2 \le t \le T;\ 1 \le j \le N$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right], 2 \le t \le T;\ 1 \le j \le N$$

Заключение:

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$

$$q_T{}^* = \operatorname*{argmax}_{1 \le i \le N} \delta_T(i)$$

Путь:

$$q_t{}^* = \psi_{t+1}(q_{t+1}^*), t = T-1, T-2, \dots, 1$$

3. Дана тренировочная последовательность. Задача состоит в том, чтобы подобрать параметры модели так, чтобы наилучшим образом описать наблюдаемую последовательность, то есть максимизировать вероятность появления наблюдаемой последовательности в данной модели.

Решение:

Аналитического решения этой задачи нет. Также остается неясным, как оценивать параметры модели. Будем выбирать такую модель $\lambda = (A, B, \pi)$, чтобы $P(O|\lambda)$ – локально максимальна, используя итеративные процедуры.

Определим $\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda)$.

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}$$

Так как $\gamma_t(i)$ – вероятность нахождения в состоянии S_i в момент времени t при заданных начальных данных, тогда $\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$.

$\sum_{t=1}^{T-1} \gamma_t(i)$ – ожидаемое число передвижений из S_i

$\sum_{t=1}^{T-1} \xi_t(i,j)$ – ожидаемое число передвижений из S_i в S_j

Проведем переоценку:

$\overline{\pi_\iota}$ – вероятность появления в состоянии S_i в начальный момент времени = $\gamma_1(i)$

$$\overline{a_{\iota J}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\overline{b_{\iota J}}(k) = \frac{\sum_{t=1, O_t = y_k}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

Если новые параметры точнее описывают данную последовательность, то заменяем параметры модели новыми параметрами и повторяем итерацию.

Часть 1. Задача распознавания активностей

Использование описанных сетей

Опишем, как используются нейросети, представленные выше, в данной работе.

Подходы к использованию HMM

Существуют два основных подхода к применению HMM:

Распознавание отдельных слов (word recognition)

Задача распознавания отдельных слов ставится следующим образом: имеются последовательности звуковых сигналов, каждой из которых соответствует слово. Требуется научиться распознавать слова по звуковым последовательностям, им соответствующим.

Предположим, что имеется словарь из V слов и каждое слово моделируется отдельной HMM. Для каждого слова из словаря имеется тренировочная последовательность размера K, каждый элемент которой является соответствующим представлением характеристик слова. Таким образом, для решения задачи распознавания отдельных слов необходимо сделать следующее:

1. Для каждого слова v из словаря необходимо построить $HMM\lambda^v$, то есть нужно подобрать параметры модели (π, A, B) так, чтобы вероятность появления наблюдаемой последовательности тренировочного множества для слова v была максимальна.
2. Для каждого неизвестного слова, которое необходимо распознать, считается вероятность появления наблюдаемой последовательности $O = \{O_1, O_2, \dots, O_T\}$ в каждой из возможных моделей $P(O|\lambda^v)$.
3. В качестве ответа выбирается слово, для которого соответствующая модель HMM показала наибольшую вероятность, то есть $v' = \text{argmax}_{v \in V} P(O|\lambda^v)$.

Распознавание речи (speech recognition)

Задача распознавания речи состоит в следующем: имеется размеченный набор данных, состоящий из данных и классов, им соответствующих. В каждый момент времени известна метка данных (вычисленная по данным, характеризующим текущий момент) и ее класс.

Для решения нужно построить соответствующую HMM и подобрать параметры (π, A, B). В качестве видимых состояний берутся метки данных, а в качестве скрытых – классы.

Таким образом, тренировочное множество состоит из пар последовательностей: $\{(X_i, O_i)|X_i = \left(x_1^i, x_2^i, \dots, x_T^i\right), O_i = (o_1^i, o_2^i, \dots, o_T^i)\}$.

Так как из тренировочного множества известны и скрытые, и видимые состояния, то оценка параметров может быть произведена непосредственным подсчетом вероятности появления видимых меток из текущих скрытых состояний, вероятностей переходов между скрытыми состояниями, а также вероятностей появления каждого класса в качестве начального состояния.

Использование HMM в данной работе

В данной работе произведена попытка использования обоих походов для распознавания активностей человека.

Чтобы применить второй подход, нужно получить по данным их метки так, чтобы в каждый момент времени исходные данные были охарактеризованы только одной целочисленной меткой (в данном наборе данных известны *x*,*y*,*z*-координаты в каждый момент времени), так как меток должно быть конечное число. Попытка использовать все три координаты повышает размерность и не позволяет варьировать и оценивать другие параметры (например, окно). Поэтому для построения целочисленных меток было решено использовать только *z*-координату.

Связь автоэнкодера и классификатора

Чтобы получить эффективный классификатор, нужно предоставить алгоритму обучения довольно большой объем данных. Однако использование большого числа маркированных данных для обучения с учителем не всегда возможно. Поэтому нужно научиться использовать немаркированные данные для самообучения нейронных сетей.

В немаркированных данных содержится меньше информации для обучения, однако объем доступных немаркированных данных заметно больше. Таким образом, мы можем подавать на вход автоэнкодера большое количество неразмеченных данных, из которых автоэнкодер будет учиться извлекать полезные признаки. Эти признаки, в свою очередь, будут использоваться для обучения классификатора с помощью относительно небольшого количества размеченных данных для обучения.

Реализация

Для реализации описанной выше модели было принято решение использовать язык программирования python и одну из его библиотек theano, которая позволяет эффективно вычислять математические выражения, включающие многомерные массивы. При помощи этого пакета реализованы нейронные сети глубокого обучения (deep learning neural networks). В работе используется модель стекового шумоподавляющего автоэнкодера. Для его написания необходимо также реализовать модели многослойного персептрона и простого шумоподавляющего автоэнкодера. Кроме того, чтобы сеть выполняла поставленную задачу, нужно добавить некую надстройку (классификатор) к верхнему уровню. На первом этапе в качестве такого классификатора использовалась логистическая регрессия. На следующем этапах регрессия была заменена на HMM, которая была реализована в двух вариантах, описанных выше. Для минимизации функции стоимости используется алгоритм градиентного спуска. Для данной задачи имеет смысл учитывать и показания датчиков не только в данный момент

времени, но и в предшествующие. Поэтому чтение данных происходит с помощью окон.

Эксперименты

Исходные данные

Тренировочное множество представляет собой высокочастотные данные (100Hz), записанные с помощью наручных регистраторов данных у 42 спящих пациентов вместе с данными полисомнографа (открытый источник [1]). Данные состоят из следующих полей:

- Метка времени
- Длина последовательности, то есть насколько часто появляются значения прибора
- 3D-значения акселерометра
- Показания светового датчика
- Фаза сна, определенная акселерометром

Подбор параметров стекового шумоподавляющего автоэнкодера

Сравнение стохастического градиентного спуска и метода сопряженных градиентов

Проведены эксперименты для сравнения работы стохастического градиентного спуска и метода сопряженных градиентов. В модели стекового шумоподавляющего автоэнкодера на стадии pre-training наблюдаются следующие результаты:

- Для первого автоэнкодера (первый слой стекового шумоподавляющего автоэнкодера) метод сопряженных градиентов сходится к минимуму заметно быстрее, чем стохастический градиентный спуск (рис. 4, 5). Он находит минимум за несколько эпох (3-4), что в 3-5 раз быстрее, чем делает это стохастический градиентный спуск при аналогичных параметрах. Однако и во втором случае время достижения минимума

оказывается довольно небольшим: не более 15 эпох. Вследствие вышесказанного, для данной задачи и имеющихся данных можем считать преимущество по скорости метода сопряженных градиентов перед стохастическим градиентным спуском на первом автоэнкодере несущественным.

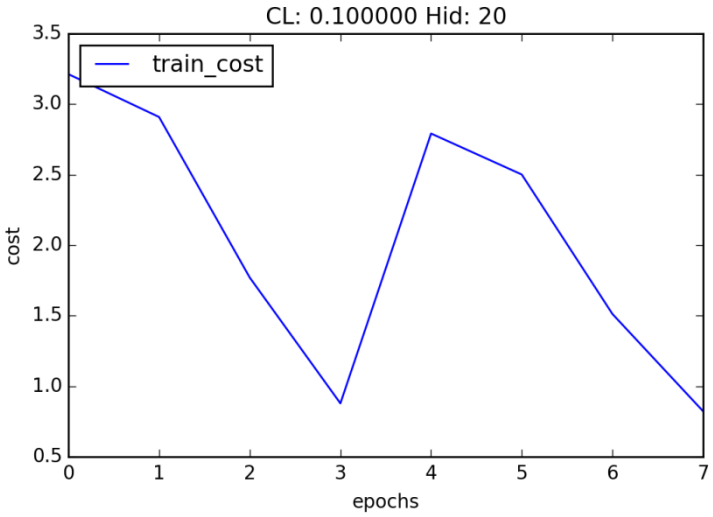


Рис. 4.

Тренировка первого слоя стекового шумоподавляющего автоэнкодера методом сопряженных градиентов.

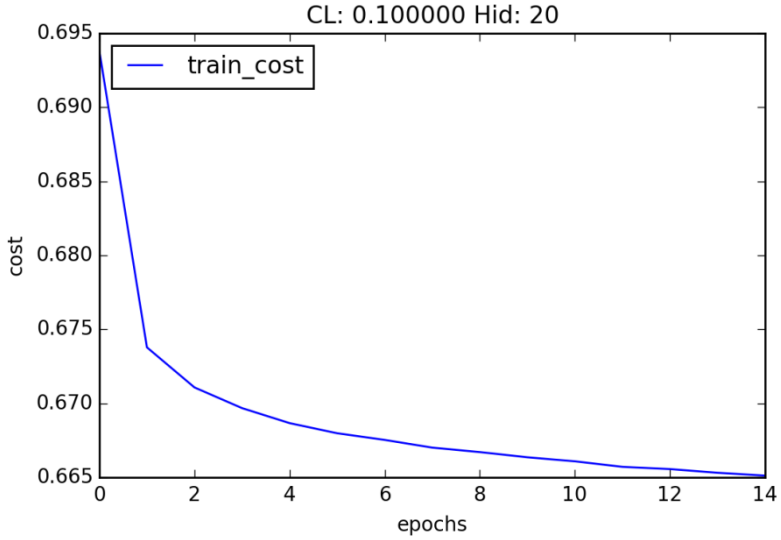


Рис. 5.

Тренировка первого слоя стекового шумоподавляющего автоэнкодера методом стохастического градиентного спуска.

- Для второго автоэнкодера (второй слой стекового шумоподавляющего автоэнкодера) метод сопряженных градиентов работает хуже: он сходится к минимуму либо за время, которое необходимо для сходимости метода стохастического градиентного спуска, либо за большее (рис. 6, 7). При этом не во всех экспериментах глобальный минимум был достигнут. Предполагается, что алгоритм закончил работу, остановившись в локальном минимуме.

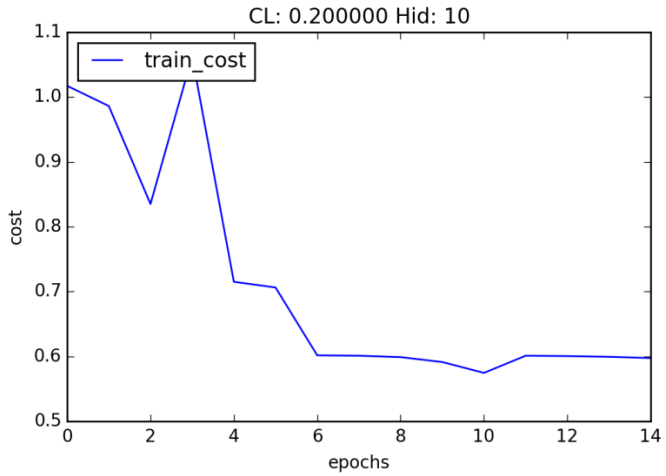


Рис. 6.

Тренировка второго слоя стекового шумоподавляющего автоэнкодера методом сопряженных градиентов.

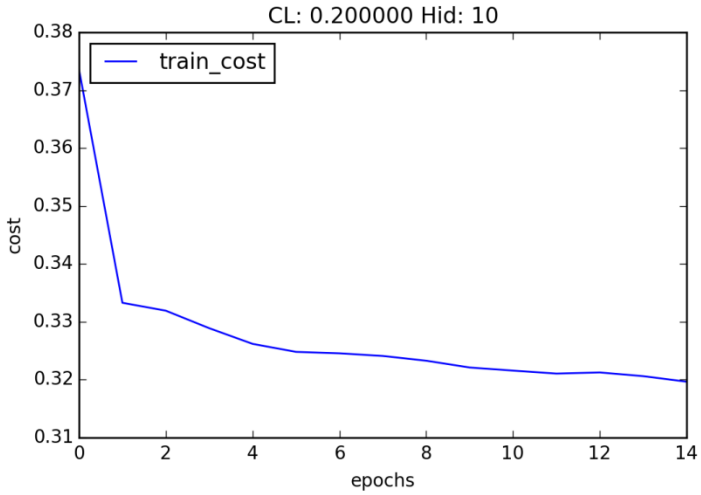


Рис. 7.

Тренировка второго слоя стекового шумоподавляющего автоэнкодера методом стохастического градиентного спуска.

То есть, для тренировки модели стекового автоэнкодера лучше работает метод стохастического градиентного спуска. Для первого автоэнкодера метод сопряженных градиентов работает быстрее, но не значительно, тогда как для второго автоэнкодера при тренировке данным методом результат гораздо хуже полученного при тренировке стохастическим градиентным спуском не только по времени, но и по качеству тренировки.

При использовании метода сопряженных градиентов следует учитывать некоторые рекомендации по его применению:

- Функция стоимости имеет единственный глобальный минимум при отсутствии локальных.
- Функция стоимости (хотя бы локально) может быть аппроксимирована квадратичной функцией.
- Функция стоимости непрерывна и имеет непрерывный градиент.
- Функция, возвращающая градиент функции стоимости, не очень велика (например, ее норма менее 1000).
- Начальное предположение об искомом векторе должно быть достаточно близко к искомому глобальному минимуму.

Таким образом, уже на стадии pre-training взаимосвязи, которые необходимо выявить для подбора оптимальных параметров второго слоя стекового шумоподавляющего автоэнкодера, оказываются слишком сложны для выявления их методом сопряженных градиентов. В данной работе этот алгоритм может быть применен для ускорения обучения лишь при выявлении не очень сложных взаимосвязей: при тренировке логистической регрессии, на стадии pre-training при тренировке первого слоя стекового автоэнкодера.

Анализ влияния размера окна на результат

Рассмотрим результаты, получившиеся при тренировке стекового шумоподавляющего автоэнкодера.

Характер поведения функции стоимости на стадии pre-training при обучении первого автоэнкодера не меняется в зависимости от размера окна (рис. 8, 9).

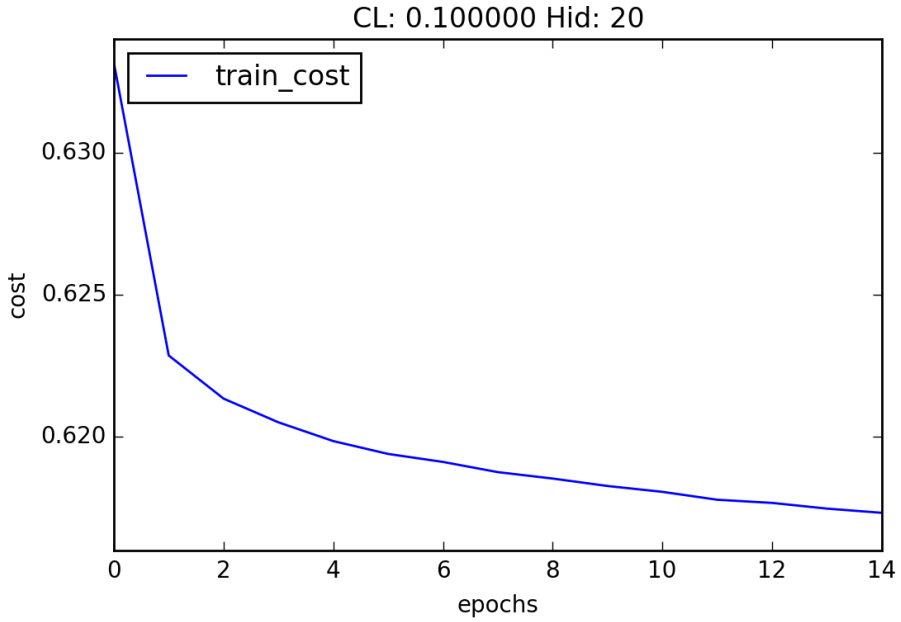


Рис. 8.

Функция стоимости первого слоя стекового шумоподавляющего автоэнкодера с учетом окна размера 10.

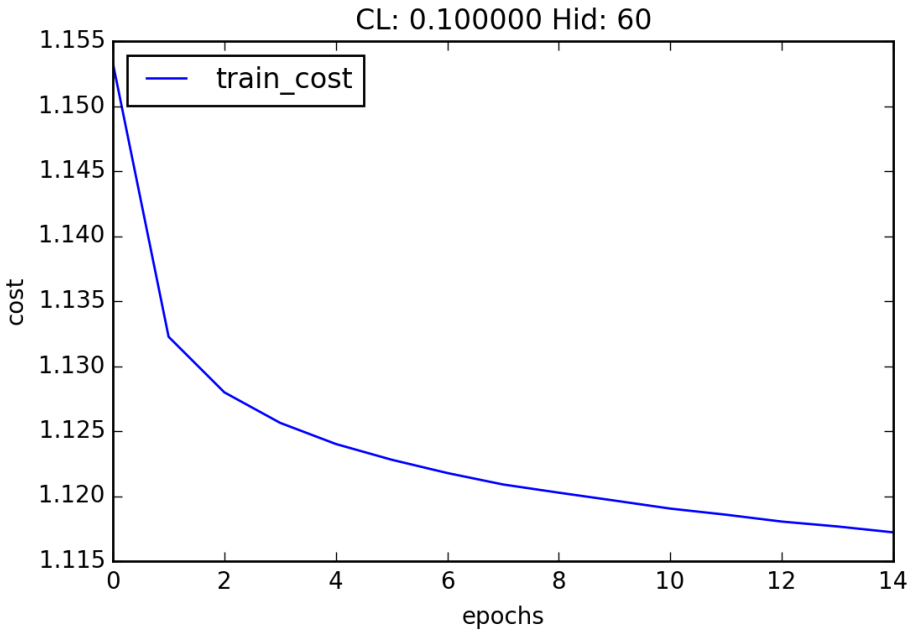


Рис. 9.

Функция стоимости первого слоя стекового шумоподавляющего автоэнкодера с учетом окна размера 30.

Такой же вывод можно сделать, исходя из графиков для второго слоя стекового автоэнкодера (рис. 10, 11).

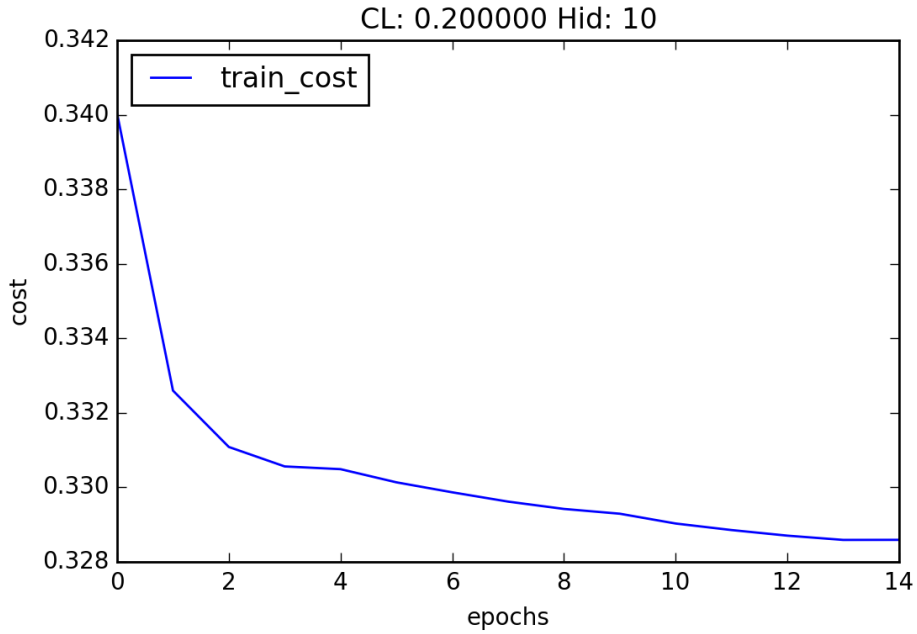


Рис. 10.

Функция стоимости второго слоя стекового шумоподавляющего автоэнкодера с учетом окна размера 10.

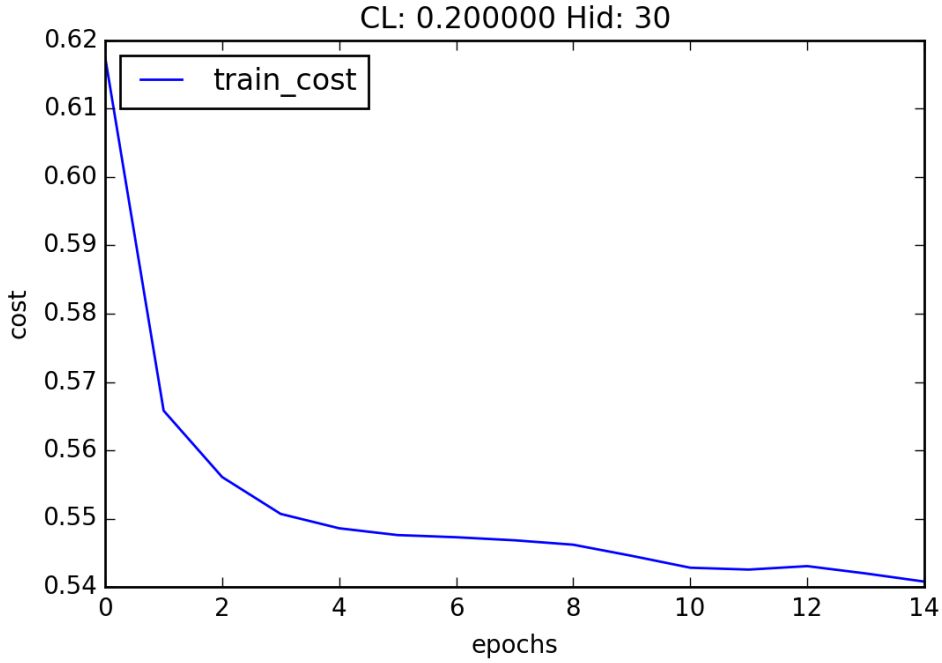


Рис.11.

Функция стоимости второго слоя стекового шумоподавляющего автоэнкодера с учетом окна размера 30.

Характер тренировки изменялся незначительно и на стадии fine-tuning (для логистической регрессии в качестве классификатора) (рис. 12, 13). Здесь скорость обучения выбиралась довольно малой (0.0001) при большом числе итераций для получения наиболее точных результатов.

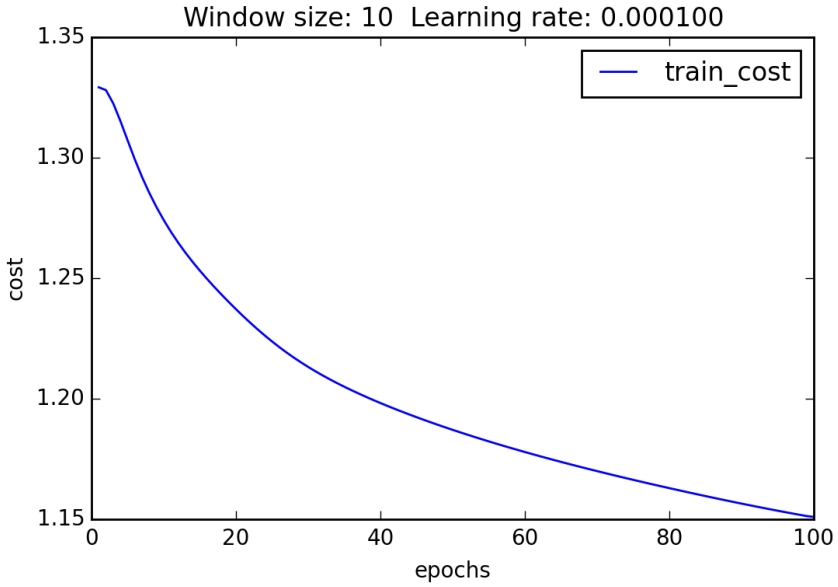


Рис.12.

Функция стоимости на стадии fine-tuning стекового шумоподавляющего автоэнкодера с учетом окна размера 10.

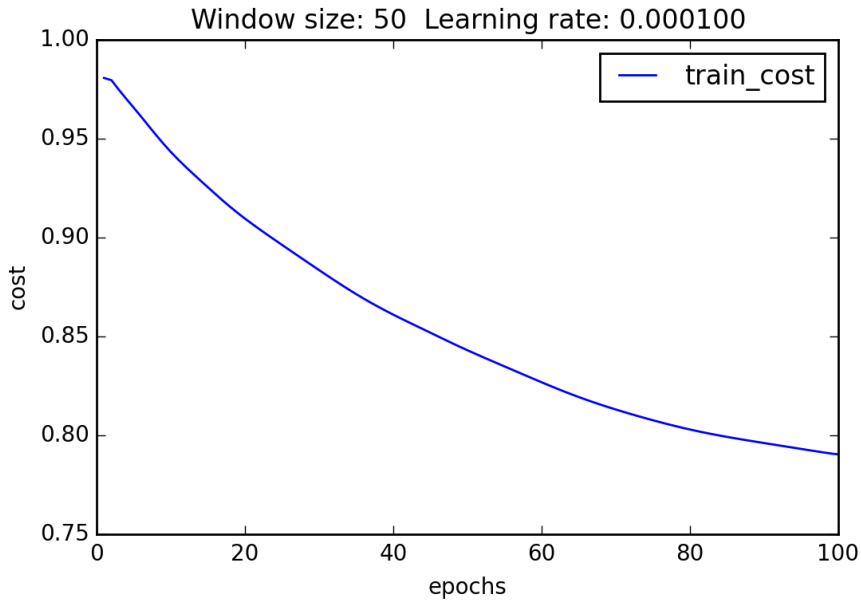


Рис. 13.

Функция стоимости на стадии fine-tuning стекового шумоподавляющего автоэнкодера с учетом окна размера 50.

Порядок ошибки на тренировочном, тестовом и валидирующем множествах при изменении окна меняется незначительно для модели стекового шумоподавляющего автоэнкодера. Небольшим преимуществом по сравнению с другими моделями обладают модели с окнами размера 20 - 100.

Логистическая регрессия

Модель классификатора

Был проведен ряд экспериментов с различными параметрами. В ходе экспериментов варьировались следующие параметры:

- Скорость обучения
- Размер окна
- Количество файлов, использованных для тренировки, валидации и тестирования (каждый файл содержит данные, характеризующие одного пациента)

Обучение прекращается, как только модель пройдет через все тренировочные примеры заданное число раз. Один проход через все тренировочные примеры будем далее называть эпохой. Однако, как показывают эксперименты, с некоторого момента ошибка на множестве для валидации перестает меняться и уже не имеет смысла продолжать обучение. Поэтому было введено прерывание обучения, если модель не меняется в течение некоторого заранее обозначенного времени.

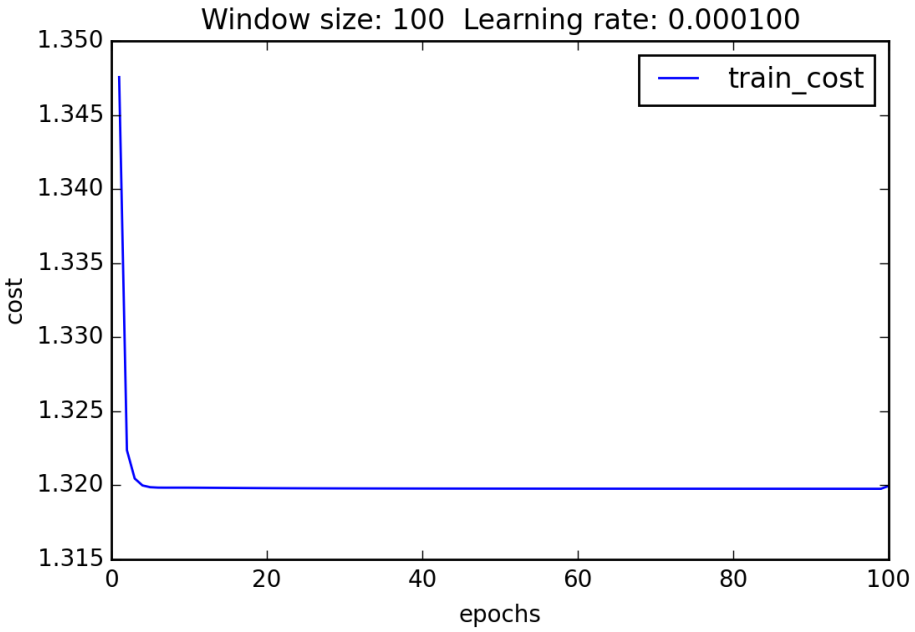


Рис. 14.

Функция стоимости логистической регрессии.

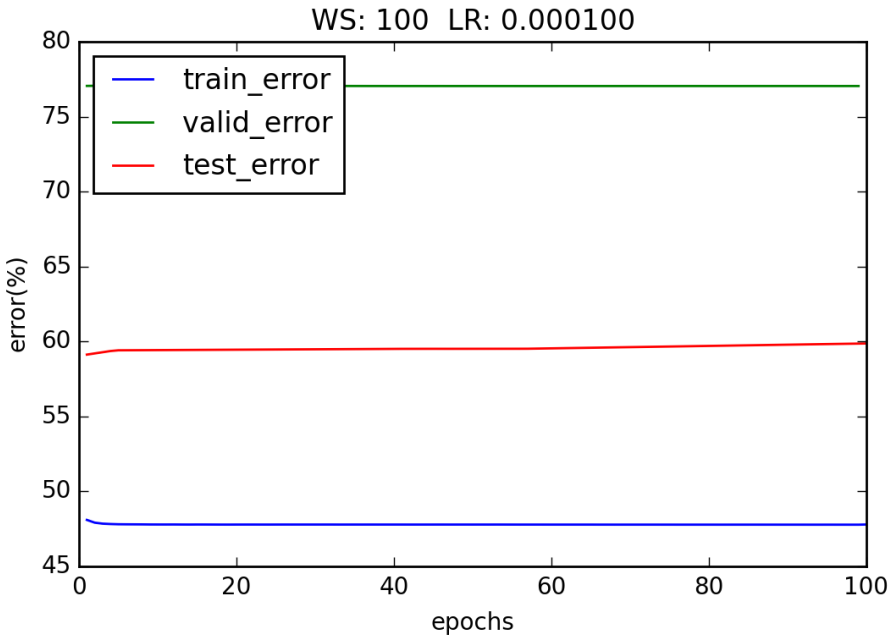


Рис. 15.

Функция ошибки логистической регрессии.

Модель стекового шумоподавляющего автоэнкодера с классификатором на вернем слое

При реализации стекового автоэнкодера и логистической регрессии получаем следующие графики (рис. 16, 17).

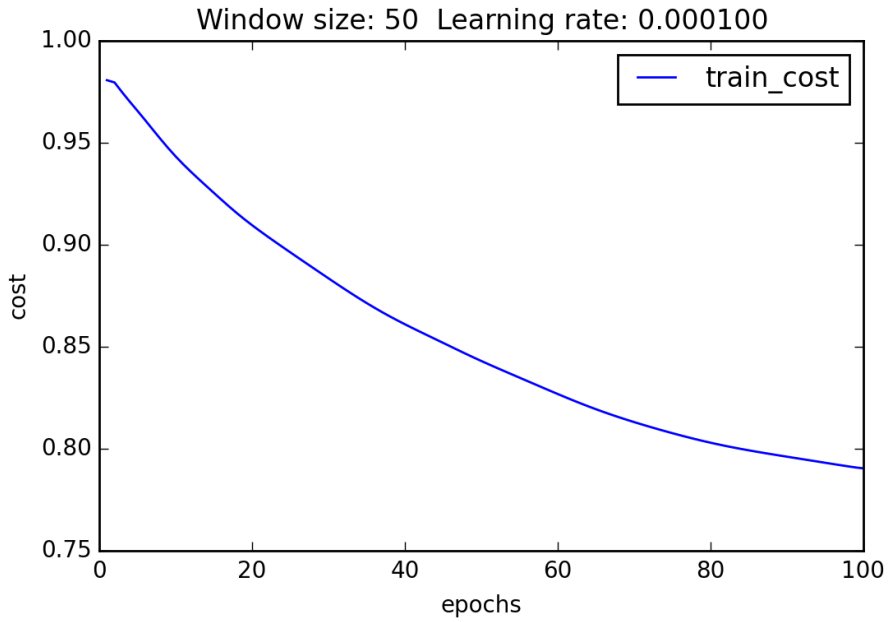


Рис. 16.

Функция стоимости для логистической регрессии на стадии fine-tuning.

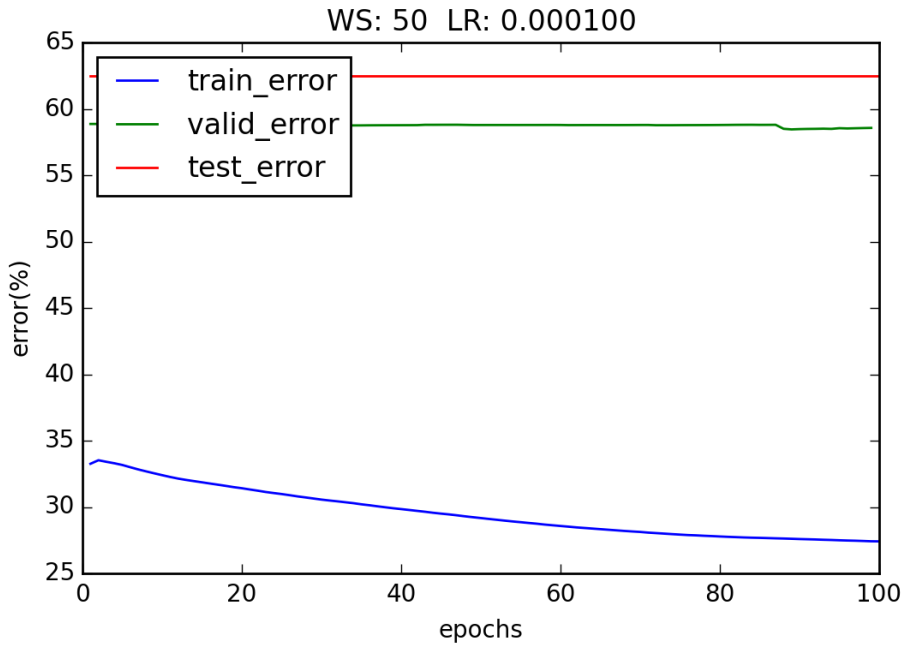


Рис. 17.

Функция ошибки для логистической регрессии на стадии fine-tuning.

HMM

Подготовка данных

Модель HMM накладывает некоторые ограничения на область ее применения. Одно из них – число скрытых и видимых состояний должно быть конечно. Однако множество значений входных данных никак не ограничено, поэтому для применения данной модели нужно подобрать отображение из множества имеющихся данных в некоторое конечное множество. При этом нужно учитывать, что от количества возможных скрытых и видимых состояний зависят размеры матриц переходов и векторов смещения. Таким образом, с ростом числа скрытых и видимых состояний количество параметров модели, которые должны быть подобраны в процессе обучения, растет экспоненциально. Здесь нужно найти оптимальное соотношение между количеством скрытых состояний (и, как следствие, количеством параметров модели) и точностью предсказаний. В связи с этим, для обучения сети используется только *z*-координата.

С целью избавления от шумов, исходная последовательность пропускается через фильтр, параметры которого также были подобраны экспериментально. Чтобы учитывать не только текущий момент времени, но и предшествующие (последующие), используются окна. Таким образом, исходные данные представляют собой не единственное значение, а последовательность значений. При реализации HMM требуется, чтобы на вход сети подавалась единственная метка, поэтому исходная последовательность значений должна быть преобразована в одно число с помощью некоторой функции. В наших экспериментах использовались две функции:

1. Среднее значение последовательности
2. Функция, учитывающая как среднее значение, так и дисперсию последовательности

Подбор параметров и результаты экспериментов

Применение HMM возможно с точки зрения двух подходов: “распознавание слов” и “распознавание речи”. Все численные показатели, приведенные ниже, показывают результаты экспериментов для второго подхода. Аналогичные эксперименты проводились и для подхода “распознавание слов”, однако точность распознавания оказывалась меньше при одинаковых значениях параметров. Поэтому в этом разделе будем описывать результаты, полученные для подхода “распознавание речи”.

Для оценки количества состояний и сравнения двух описанные выше функций, сопоставляющих последовательности чисел одно число, были проведены соответствующие эксперименты. Результаты и соответствующие выводы приведены ниже. Использованы следующие обозначения:

- Train_set – множество данных, на которых модель обучается
- Test_set – множество для тестирования качества модели
- Preprocess – преобразование входных данных
- Window – размер окна
- Rank – количество значащих цифр
- Mean_error – среднее значение ошибки на тестовом множестве
- Normalize [-1; 1] – отображение входных данных в отрезок [-1; 1]
- Avg – использование среднего значения по окну вместо учета всех значений окна
- Avg_disp – использование среднего значения и дисперсии вместо учета всех значений окна

train_set	test_set	preprocess	window	rank	mean_error
4 файла	all_data\train_set	normalize [-1; 1]	1	1	0.6948460
4 файла	all_data\train_set	normalize [-1; 1]	1	3	0.6124235
4 файла	all_data\train_set	normalize [-1; 1]	1	5	0.5765520
4 файла	all_data\train_set	normalize [-1; 1]	1	6	0.5765520
4 файла	all_data\train_set	avg	10	5	0.5296880

4 файла	all_data\train_set	avg	20	5	0.5111960
4 файла	all_data\train_set	avg	30	5	0.5163930
4 файла	all_data\train_set	avg	50	5	0.5223300
4 файла	all_data\train_set	avg	100	5	0.5215360
4 файла	all_data\train_set	avg	100	6	0.5339520

Вывод: Увеличение точности входных данных (нормализованных) до шестого знака не улучшает результат по сравнению с использованием первых пяти знаков после запятой, при этом повышается размерность. Вследствие этого в дальнейших экспериментах входные данные брались с точностью, не превышающей пять знаков после запятой.

4 файла	all_data\train_set	avg_disp	30	5	0.5172500
4 файла	all_data\train_set	avg_disp	50	5	0.5238440
4 файла	all_data\train_set	avg_disp	100	5	0.5380460
4 файла	all_data\train_set	avg_disp	300	5	0.5698030
4 файла	all_data\train_set	avg_disp	600	5	0.5956400

Вывод: Добавление дисперсии к среднему значению не улучшает точность, но повышает размерность входных данных. При этом при увеличении окна точность предсказания модели падает.

Далее были проведены эксперименты с фильтром. Результаты и соответствующие выводы приведены ниже. В дополнение к описанным выше обозначениям использованы следующие:

- Filter_order – порядок фильтра
- Filter+avg – использование среднего значения предварительно отфильтрованных данных по окну вместо учета всех значений окна
- Filter+avg_disp – использование среднего значения и дисперсии предварительно отфильтрованных данных вместо учета всех значений окна

train_set	test_set	preprocess	window	rank	filter_order	mean_error
4 файла	all_data\train_set	filter+avg	20	5	3	0.5180700
4 файла	all_data\train_set	filter+avg	20	5	4	0.5175620
4 файла	all_data\train_set	filter+avg	30	5	3	0.5269000
4 файла	all_data\train_set	filter+avg	30	5	4	0.5246360
4 файла	all_data\train_set	filter+avg	50	5	3	0.5299220
4 файла	all_data\train_set	filter+avg	50	5	4	0.5268510
4 файла	all_data\train_set	filter+avg	100	5	3	0.5264380
4 файла	all_data\train_set	filter+avg	100	5	4	0.5255550
4 файла	all_data\train_set	filter+avg	300	5	3	0.5567700
4 файла	all_data\train_set	filter+avg	300	5	4	0.5500490

Вывод: Использование среднего значения по фильтрованным данным не улучшает результат по сравнению со средним без предварительной фильтрации для любых размеров окон. При этом размерность в обоих случаях одинакова. Использование фильтров 3 и 4 порядков дает практически одинаковый результат.

4 файла	all_data\train_set	filter+avg_disp	30	5	3	0.5707120
4 файла	all_data\train_set	filter+avg_disp	30	5	4	0.5706700
4 файла	all_data\train_set	filter+avg_disp	300	5	3	0.5830240
4 файла	all_data\train_set	filter+avg_disp	300	5	4	0.5824670

Вывод: Использование дисперсии вместе со средним значением окна после фильтра не дает дополнительных преимуществ по сравнению с использованием среднего по фильтрованным данным вне зависимости от размера окна, при этом увеличивается размерность данных. Порядок фильтра (3 или 4), как и в предыдущем случае, не влияет на точность предсказания. Таким образом, использование дисперсии во всех экспериментах повышает размерность исходных данных, ухудшая результат. В дальнейших экспериментах дисперсия не используется.

Кроме того, результаты экспериментов показывают, что при использовании

больших окон распознавание происходит несколько хуже. Это может быть связано с используемыми функциями преобразования последовательностей в одно значение: при больших размерах окон может быть потеряна существенная часть информации.

Дальнейшие эксперименты в том же ключе позволили сделать следующий вывод: результат работы фильтра напрямую связан с используемым окном. При больших окнах две описанные выше функции хорошо справляются с шумами, и фильтр лишь ухудшает результат. При малых размерах окон экспериментальные результаты с фильтром и без него сравнимы, кроме случая тестирования на тренировочных данных: в этом случае применение фильтра существенно улучшает качество распознавания (до 30% по сравнению с экспериментами без фильтра).

Таким образом, имеет смысл использовать фильтр с небольшими размерами окон. Это утверждение подтверждает, например, следующий эксперимент: возьмем в качестве тренировочного множества все имеющиеся файлы, кроме одного. Пропущенный файл будем использовать в качестве тестового множества, то есть будем считать ошибку работы модели на основе данных из этого файла. Поступим так с каждым файлом из имеющегося набора, а затем посчитаем среднюю ошибку по все файлам. В данном эксперименте средняя ошибка, полученная при использовании среднего по окну размера 20, равна 55.774 %, тогда как при тех же параметрах, но с предварительной фильтрацией данных – 53.4165 % . Описание параметров и результаты нескольких экспериментов по вышеописанной схеме представлены ниже:

1. В качестве входных данных берем среднее по окну размера 20, предварительно пропустив данные через фильтр третьего порядка, учитывая пять значащих цифр. Среднее значение ошибки по всем имеющимся данным 53.42 %.

Test_file	1	2	3	4	5	6	7	8	9
Mean_error	47.13	62.01	39.80	38.67	28.41	51.92	69.67	76.27	70.18

Test_file	10	11	12	13	14	15	16	17	18
Mean_error	77.50	76.43	49.57	50.98	70.12	61.36	35.52	46.90	39.36

Test_file	19	20	21	22	23	24	25	26	27
Mean_error	59.39	53.22	50.57	61.41	39.58	28.35	30.32	55.61	61.28

Test_file	28	29	30	31	32	33	34	35	36
Mean_error	49.27	55.37	52.12	74.00	51.74	59.56	35.52	47.44	60.93

Test_file	37	38	39	40	41	42	43	44	45
Mean_error	80.91	65.19	44.40	54.91	65.06	57.88	49.09	50.00	21.84

Test_file	46
Mean_error	50.40

2. В качестве входных данных берем среднее по окну размера 20, учитывая пять значащих цифр. Среднее значение ошибки по всем имеющимся данным 55.77 %.

Test_file	1	2	3	4	5	6	7	8	9
Mean_error	40.11	50.25	37.89	52.41	36.00	65.50	56.87	65.48	70.48

Test_file	10	11	12	13	14	15	16	17	18
Mean_error	94.43	74.39	74.16	53.17	66.36	40.73	64.58	46.59	43.96

Test_file	19	20	21	22	23	24	25	26	27
Mean_error	66.22	58.20	58.37	63.13	37.68	37.93	39.27	53.02	60.00

Test_file	28	29	30	31	32	33	34	35	36
Mean_error	80.72	51.06	54.94	65.43	48.77	60.21	46.11	46.76	56.94

Test_file	37	38	39	40	41	42	43	44	45
Mean_error	64.03	58.71	44.02	58.71	67.04	59.27	54.02	41.02	31.19

Test_file	46
Mean_error	69.49

3. В качестве входных данных берем среднее по окну размера 60, учитывая пять значащих цифр. Среднее значение ошибки по всем имеющимся данным 55.26 %.

Test_file	1	2	3	4	5	6	7	8	9
Mean_error	43.13	49.27	39.04	46.79	33.76	63.10	59.38	76.29	72.53

Test_file	10	11	12	13	14	15	16	17	18
Mean_error	90.63	78.07	69.55	56.07	68.29	49.94	35.57	49.89	40.78

Test_file	19	20	21	22	23	24	25	26	27
Mean_error	66.13	55.24	59.34	58.39	44.70	34.01	34.49	61.16	62.51

Test_file	28	29	30	31	32	33	34	35	36
Mean_error	64.53	58.86	55.08	66.25	49.22	54.86	45.14	47.36	56.29

Test_file	37	38	39	40	41	42	43	44	45
Mean_error	72.35	62.97	44.64	55.23	71.36	55.19	50.59	40.41	38.14

Test_file	46
Mean_error	55.79

Наименьшая ошибка классификатора, полученная в ходе экспериментов, составляет 53,4 %. При встраивании HMM в стековый шумоподавляющий автоэнкодер в качестве верхнего уровня (уровня классификатора) средняя ошибка на тех же параметрах достигает 51,1 %.

Выводы по задаче распознавания активностей

Проведена работа по изучению нейронных сетей. Особое внимание уделено следующим типам сетей: логистическая регрессия, многослойный персептрон, автоэнкодеры (в том числе шумоподавляющий и стековый шумоподавляющий), скрытые марковские модели. Построены три модели (реализация на Python), каждая из которых представляет собой последовательное объединение стекового автоэнкодера и классификатора в единую структуру. Для моделей был проведен ряд экспериментов с целью подбора оптимальных параметров.

Однако эксперименты с данными моделями не дают достаточно точный результат. Было сделано предположение о том, что причина достаточно большой ошибки состоит в том, что во время обучения функция стоимости не достигает глобального минимума, а останавливается в локальном. В связи с этим предположением возникает идея о применении альтернативного подхода к обучению, основной смысл которого состоит в том, чтобы обучать нейросеть не один раз, а несколько, каждый раз начиная со случайной инициализации параметров сети и выбирая лучший образец. Такой подход не гарантирует нахождение глобального минимума. Однако благодаря случайной инициализации для каждой новой модели, мы получаем возможность выбрать наиболее подходящую для данной задачи модель из имеющихся.

Таким образом, возникает задача сравнения стандартного подхода к обучению нейросетей (одна нейросеть, обучаемая длительное время) и подхода с несколькими случайными инициализациями, обучаемыми достаточно короткое время. Однако для сравнения подходов к обучению использовать модели стекового шумоподавляющего автоэнкодера в связке с HMM не имеет смысла, так как для HMM такое сравнение не актуально:

- В методе “распознавание речи” нет обучения как такового: параметры модели считаются непосредственно исходя из тренировочного

множества, поэтому мы не можем сравнить эти подходы к обучению нейросетей.

- В методе “распознавание слов” мы обучаем одновременно сразу несколько моделей (по одной для каждого слова из словаря). Попытка применить данный метод для сравнения подходов к обучению сетей не дает однозначного результата, так как классификация неизвестного для модели слова зависит от работы сразу нескольких моделей.

Вследствие вышесказанного, для сравнения двух описанных подходов к обучению нейросети, было решено использовать стековый шумоподавляющий автоэнкодер с логистической регрессией в качестве классификатора.

Часть 2. Задача распознавания речи

Постановка задачи

Имеется набор данных, представляющих собой пофонемно размеченные аудиозаписи речи. Необходимо создать модель классификатора, которая по аналогичным данным сможет определять фонему, соответствующую звуковой последовательности, и сравнить на этой модели подходы к обучению нейросетей:

1. Стандартный: длительное обучение одной модели.
2. Соревновательный: обучение нескольких моделей в течение небольшого числа эпох (при этом каждый раз обучение начинается со случайной инициализации, не зависящей от результатов тренировки предыдущих моделей).

При оценке моделей должны учитываться качество распознавания и скорость обучения.

Модель нейросети и методы ее обучения

Подходы к обучению нейросетей

Опишем подробнее, какие подходы к обучению нейросети могут быть применены:

Стандартный подход

Подход, который обычно применяется при тренировке нейросетей, состоит в следующем: веса инициализируются единожды, затем сеть обучается до тех пор, пока не будет выполнено условие прекращения работы алгоритма. При этом есть вероятность остановиться в локальном минимуме функции стоимости. В этом случае в результате обучения будут подобраны параметры, при которых функция стоимости достигает не глобальный, а локальный минимум (который может оказаться достаточно далеко от

искомого глобального), что отразится и на итоговой точности предсказаний модели.

WHILE NOT *выполнено_условие_остановки*:

    Вычислить_ошибку_и_переинициализировать_веса

ENDWHILE

Соревновательный подход

Может быть рассмотрен альтернативный подход: инициализируем веса, обучаем сеть некоторое (не очень длительное) время, запоминаем полученный результат и затем начинаем процесс сначала. Таким образом, на выходе работы алгоритма имеем несколько различных моделей, обучавшихся не очень долгое время. Далее эти модели оцениваются, результаты сравниваются и из всех полученных таким образом результатов выбирается наилучший (можно выбирать лучшую модель и в процессе обучения, чтобы не хранить лишние данные о заведомо более плохих моделях). Так как каждый раз инициализация параметров случайна и не зависит от предыдущей модели, то при попадании модели в некий локальный минимум, следующая модель может не попасть в него. Это не исключает того, что новая модель попадет в другой (или тот же) локальным минимум, и, в конечном счете, ни в одной из моделей глобальный минимум достигнут не будет. Тогда мы получаем возможность выбрать наиболее подходящий для данной задачи локальный минимум.

FOR *current_attempt*=1 to *all_attempts* do:

    WHILE NOT *выполнено_условие_прекращения*:

        Вычислить_ошибку_и_переинициализировать_веса

    ENDWHILE

    IF *current_attempt_error* < *best_attempt_error*:

        Сохранить_лучшую_модель

    ENDIF

ENDFOR

Гибрид

Этот вариант является комбинацией двух предыдущих: сначала применяем второй подход (то есть проводим не очень длительное обучение несколько раз для различных инициализаций) с сохранением параметров нейросети, для которой был получен наилучший результат, а затем учим сохраненную сеть до выполнения некоторого, выбранного нами, критерия остановки.

FOR *current_attempt*=1 to *all_attempts* do:

WHILE NOT *выполнено_условие_прекращения*:

Вычислить_ошибку_и_переинициализировать_веса

ENDWHILE

IF *current_attempt_error* < *best_attempt_error*:

Сохранить_лучшую_модель

ENDIF

ENDFOR

WHILE NOT *выполнено_условие_прекращения*:

Вычислить_ошибку_и_переинициализировать_веса_в_лучшей_модели

ENDWHILE

В рамках такого подхода возможен и следующий алгоритм обучения: на первом этапе тренируется достаточно большое количество моделей в течение нескольких эпох, затем они оцениваются, сравниваются, и выбирается некоторое (заранее заданное) число лучших моделей, проходящих во второй этап. На втором этапе выбранные модели обучаются еще по несколько эпох, затем они снова оцениваются и сравниваются; лучшие модели проходят в третий этап и т.д. При этом количество этапов и количество моделей, проходящих в следующий этап – параметры алгоритма, которые могут изменяться.

Задание_массива_winners_count[all_stages]

FOR *current_stage*=1 to *all_stages* do:

IF *current_stage* > 1:

Best_models_from_previous_stage = best_models_on_current_stage

```
        ELSE:
            Best_models_from_previous_stage = all_models
        ENDIF
        best_models_on_current_stage = new array[winners_count[current_stage]]
        max_error_for_winners = infimum
        FOR current_attempt=1 to all_attempts_on_current_stage do:
            WHILE NOT выполнено_условие_прекращения:
                Вычислить_ошибку_и_переинициализировать_веса
            ENDWHILE
            IF current_attempt_error < max_error_for_winners:
                Сохранить_модель_в_массив_best_models_on_current_stage
                IF best_models_on_current_stage IS FULL:
                    Обновить_ max_error_for_winners
                ENDIF
            ENDIF
        ENDFOR
ENDFOR
```

Модель нейронной сети

Проведен ряд экспериментов с различными параметрами. В ходе экспериментов варьировались следующие параметры:

- Скорость обучения
- Количество эпох обучения
- Критерии остановки обучения
- Размер mini-batches

В работе приведены результаты для нейросети с одним скрытым слоем и 200 нейронами в нем. Сеть представляет собой стековый автоэнкодер, на верхнем уровне которого находится классификатор – логистическая регрессия. Функцией активации в скрытом слое стекового автоэнкодера является

логистическая функция. В качестве функции активации для слоя классификатора используется функция softmax. Для сравнения вышеописанных подходов к обучению нейросетей были также проведены эксперименты с простым классификатором – логистической регрессией.

В описанной выше модели менялся размер mini-batches и скорость обучения, подбирались условия прекращения обучения (одно из которых всегда было прохождение через определенное количество эпох). При этом были проведены эксперименты и с единственным критерием остановки – достижение некоторого заранее определенного количества эпох. Но при сохранении остальных параметров точность предсказания сети с единственным критерием остановки падает: при небольшом числе эпох из-за “недообучения”, при большом числе эпох – из-за переобучения на тренировочном множестве.

В экспериментах применялись следующие комбинации вышеописанных подходов:

- Pre-training
    - Длительное обучение каждого из уровней одного автоэнкодера – стандартный pre-training
    - Несколько недолгих попыток обучить автоэнкодер и выбор лучшего для каждого из уровней стекового автоэнкодера – соревновательный pre-training
- Fine-tuning: Для каждой из моделей, полученных после стадии pre-training на предыдущем шаге, проведены следующие эксперименты:
    - Длительное обучение стекового автоэнкодера как единой модели – стандартный fine-tuning
    - Несколько недолгих попыток обучить стековый автоэнкодер как единое целое и выбор лучшей нейросети – соревновательный fine-tuning

Для оценки качества экспериментов использовались следующие функции стоимости:

- Для автоэнкодера – стандартная квадратичная ошибка
- Для логистической регрессии – логарифмическая функция правдоподобия

Реализация

Для сравнения подходов к обучению нейросетей используется модель, реализованная для задачи распознавания активностей. Она представляет собой стековый шумоподавляющий автоэнкодер с логистической регрессией в качестве классификатора. Для минимизации функции стоимости используется алгоритм градиентного спуска. Имеющиеся данные уже учитывают окно, поэтому в данной задаче чтение с помощью окон нецелесообразно.

Эксперименты

Исходные данные

Исходные данные представляют собой аудиозаписи речи, которые размечены по фонемам: имеется информация о том, какая фонема произносится в каждый конкретный момент времени. На основе исходных данных строится спектрограмма с использованием преобразования Фурье; такое построение происходит с учетом окна в 25ms с шагом 10ms.

Сравнение подходов

Проведен ряд экспериментов с различными параметрами с целью сравнения подходов к обучению нейросетей. Для модели стекового автоэнкодера с логистической регрессией в качестве классификатора последовательно проводились две стадии обучения: pre-training и fine-tuning. На стадии pre-training единственный скрытый слой обучаемой модели тренировался

каждым из описанных выше способов с различными параметрами. Таким образом, после стадии pre-training имеем несколько моделей автоэнкодеров с отличными от других моделей параметрами. После стадии pre-training каждая из полученных моделей проходила fine-tuning описанными выше способами с различными скоростями обучения.

Во всех экспериментах временные затраты на тренировку соревновательным подходом в 4-5 раз ниже, чем на тренировку стандартным подходом. Поэтому, при одинаковом качестве результата, следует отдавать предпочтение соревновательному подходу.

Стадия pre-training

На стадии pre-training производилось обучение для двух размеров mini-batches, варьировалась также скорость обучения. Обучение, также как и в предыдущей задаче, прекращается, когда модель пройдет через все тренировочные примеры заданное число раз (количество эпох). Начиная с некоторого момента модель, зачастую, перестает улучшаться. В этих случаях тренировка также останавливается. Однако этот критерий прерывания обучения может быть использован только при применении стандартного подхода. При тренировке соревновательным подходом модель постоянно изменяется в процессе тренировки, так как количество тренировочных эпох невелико.

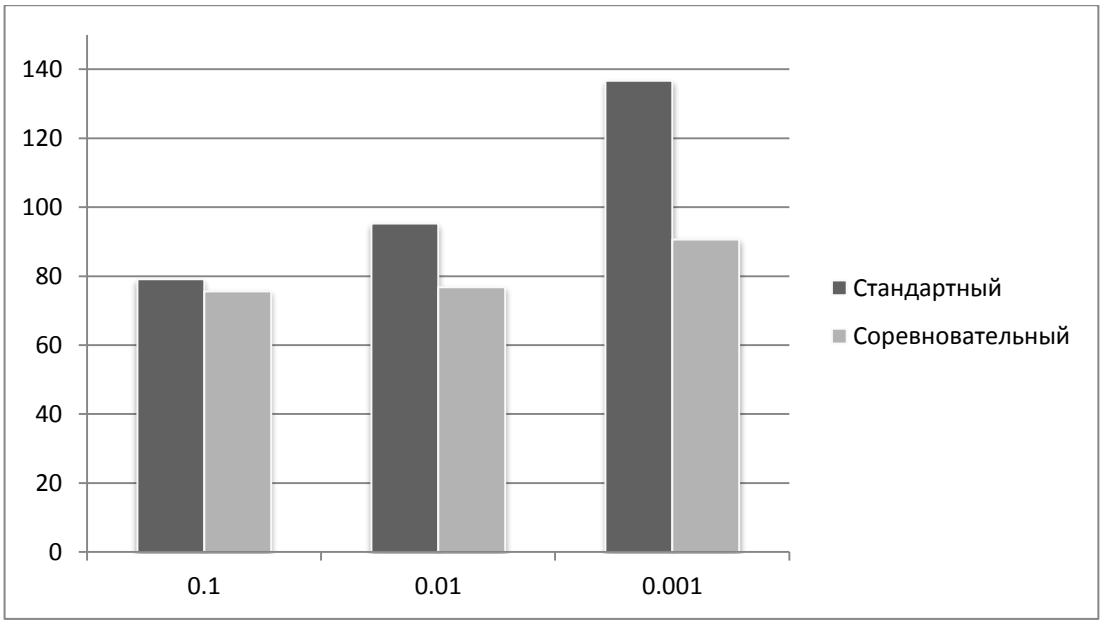
Для стандартного подхода для обоих размеров mini-batches наилучшие результаты были получены для скорости обучения порядка 0.1. При выборе меньших скоростей обучения ошибка, полученная при восстановлении искаженных входных данных, выше. Например, при снижении скорости обучения на один порядок получаем в полтора раза большую стоимость.

Соревновательный подход включает в себя следующие стадии:

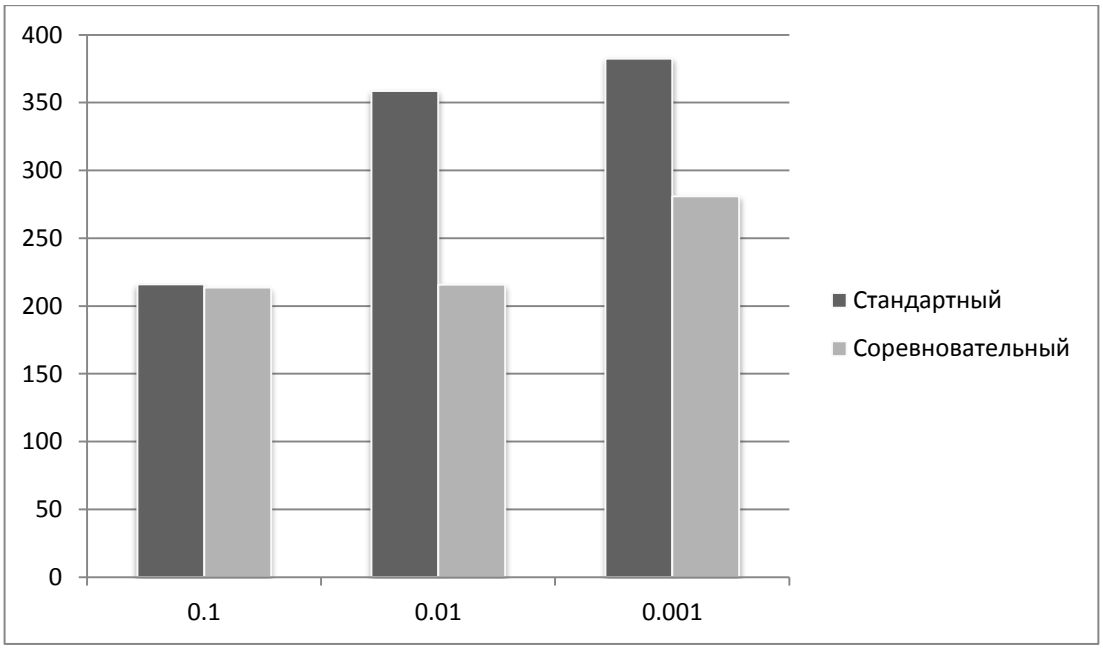
- Тренировка 15 моделей в течение 15 эпох
- Выбор лучшей модели (оценка производится с помощью функции стоимости)

Для соревновательного подхода разница в результатах при выборе различных скоростей обучения не так значительна, однако, также как и для стандартного подхода, скорость обучения порядка 0.1 оказывается оптимальной.

На диаграмме представлены значения функции стоимости для размера mini-batches 128, скоростей обучения 0.1, 0.01, 0.001 для стандартного и соревновательного подходов:



На диаграмме представлены значения функции стоимости для размера mini-batches 1000, скоростей обучения 0.1, 0.01, 0.001 для стандартного и соревновательного подходов:



Вывести зависимость результата от размера mini-batches на стадии pre-training не удается из-за специфики функции стоимости, которая зависит от размера входных данных: она растет с ростом размера mini-batches. Значение этого параметра может быть оценено только на стадии fine-tuning, когда оценивается процент ошибки классификации.

Если сравнивать соревновательный и стандартный подходы при одинаковых значениях параметров (размер mini-batches и скорость обучения), то соревновательный подход показывает лучшие результаты при всех значениях параметров с точки зрения функции стоимости, отражающей точность восстановления исходных данных. Также видно, что при использовании скорости обучения 0.1 функция стоимости для разных подходов практически одинакова.

Таким образом, в процессе pre-training наилучшие модели получаем при скорости обучения порядка 0.1.

Стадия fine-tuning

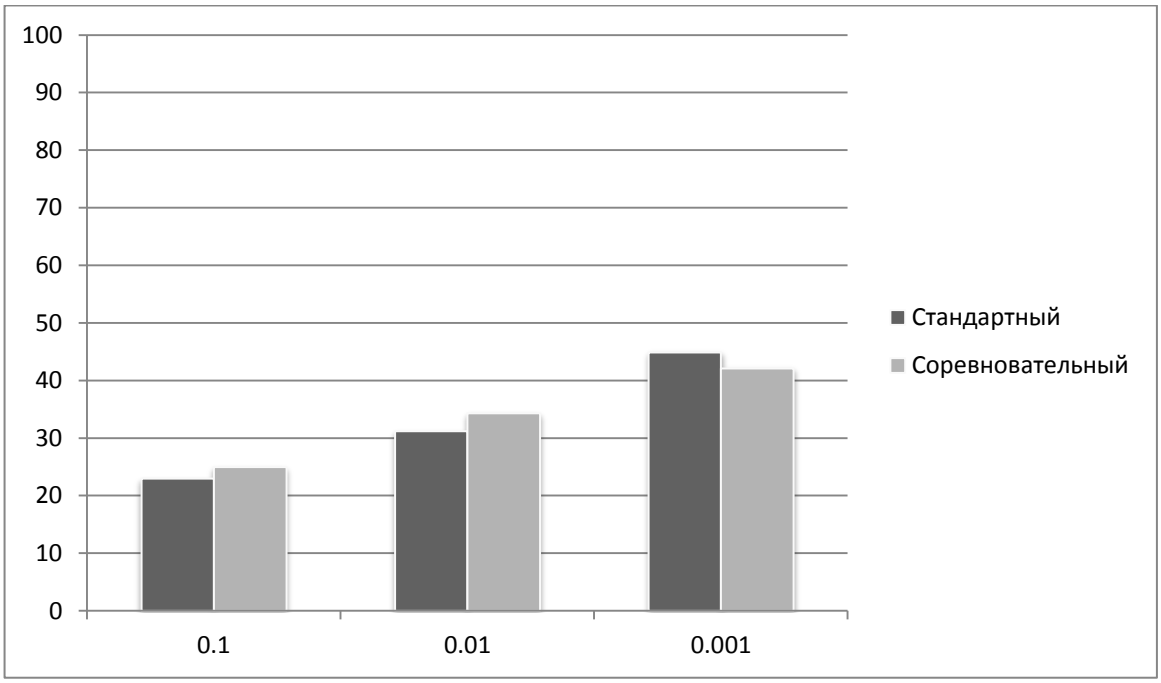
Для стадии fine-tuning были отобраны модели, показавшие наилучшие результаты после стадии pre-training, то есть модели, которые тренировались со скоростью обучения порядка 0.1. При этом были взяты и модели, обученные на стадии pre-training стандартным способом, и модели, обученные на стадии pre-training соревновательным способом. Каждая из них обучалась стандартным и соревновательным fine-tuning.

Обучение модели после стандартного pre-training

Рассмотрим модели стековых автоэнкодеров, полученных после стандартного pre-training. Обучим их двумя способами для каждого значения скорости обучения: стандартным и соревновательным. Здесь, также как и на стадии pre-training, наилучшие результаты были получены при скоростях обучения порядка 0.1 для обоих подходов. При этом, сравнивая подходы к обучению на стадии fine-tuning с одинаковыми параметрами на обеих

стадиях, получаем в среднем на 2-3% лучший результат при использовании стандартного подхода по сравнению с соревновательным.

На приведенной ниже диаграмме отражен процент ошибки тренировки для размера mini-batches 1000, скоростей обучения 0.1, 0.01, 0.001 для стандартного и соревновательного fine-tuning после стандартного pre-training со скоростью обучения 0.1:



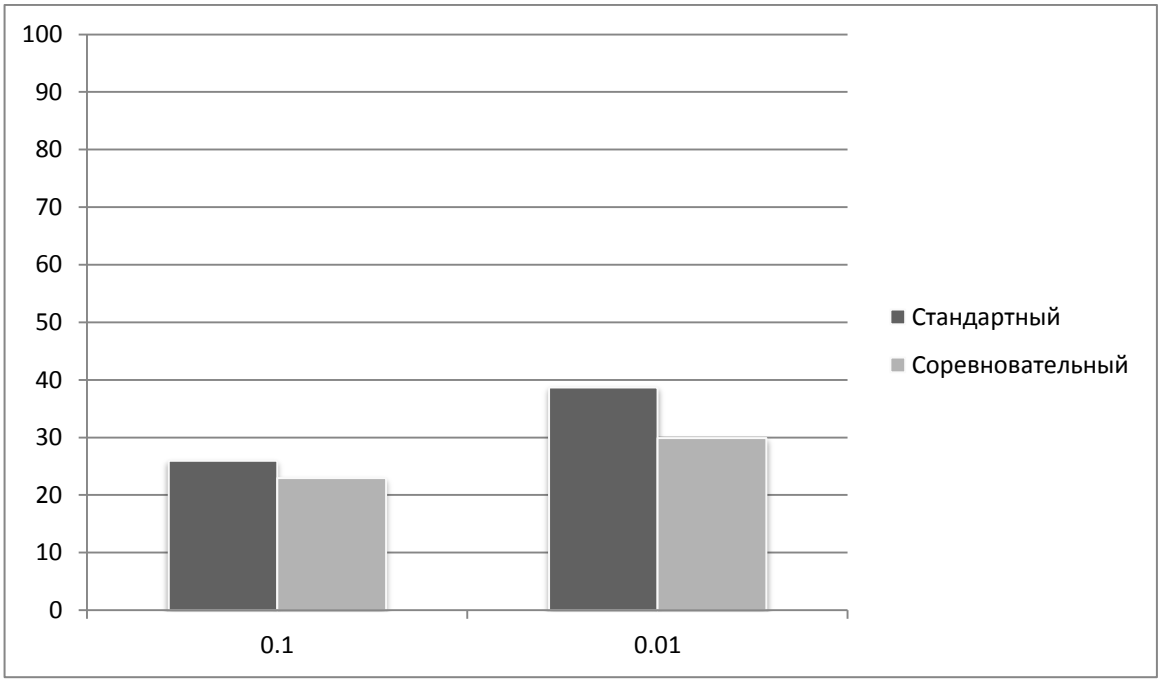
Аналогичные диаграммы получены и для других параметров.

Следовательно, для модели, обученной на стадии pre-training стандартным подходом, лучше всего продолжить обучать таким же образом с высокой скоростью обучения.

Обучение модели после соревновательного pre-training

Рассмотрим модель стекового автоэнкодера, полученную после соревновательного pre-training. Для такой модели наилучший результат получается при скорости обучения 0.3 как для стандартного, так и для соревновательного fine-tuning. При сравнении результатов тренировки стандартным и соревновательным подходом очевидно преимущество второго подхода. При его применении получаем ошибку классификации на 2-4% меньше, чем при использовании стандартного подхода.

На приведенной ниже диаграмме представлен процент ошибки тренировки для размера mini-batches 128, скоростей обучения 0.1, 0.01 для стандартного и соревновательного fine-tuning после соревновательного pre-training со скоростью обучения 0.1:



Аналогичные диаграммы получены и для других параметров.

Таким образом, модель, обученную соревновательным pre-training, следует продолжать обучать тем же способом и на стадии fine-tuning. При этом имеется преимущество не только по качеству классификации, но и по времени тренировки моделей.

Стандартный подход на стадии fine-tuning

Рассмотрим модели стекового автоэнкодера, полученные на стадии pre-training тренировкой стандартным и соревновательным подходами. Обучим эти модели на стадии fine-tuning стандартным образом. Во всех экспериментах получаем (при одинаковых параметрах) меньшую ошибку у моделей, обученных на стадии pre-training стандартным образом. При этом ошибка классификации в среднем меньше на 5-7%.

Следовательно, для обучения стандартным fine-tuning следует выбирать модели, обученные тем же образом на стадии pre-training.

Соревновательный подход на стадии fine-tuning

Рассмотрим модели стекового автоэнкодера, полученные на стадии pre-training тренировкой стандартным и соревновательным подходами. На стадии fine-tuning будем использовать соревновательный подход. Во всех экспериментах получаем схожие результаты и для моделей, обучавшихся на стадии pre-training стандартным образом, и для моделей, обучавшихся на стадии pre-training соревновательным образом. Разница в ошибке не превышает 1%, но все же модели после стандартного fine-tuning чуть точнее. Однако, при тренировке соревновательным подходом, получаем существенное преимущество по времени тренировки. При этом скорость обучения 0.3 дает наилучший результат.

Сравнение размеров mini-batches

Для выбора оптимального размера mini-batches были проведены эксперименты с разными размерами mini-batches при одинаковых скоростях и подходах к обучению. Во всех экспериментах меньшую ошибку показали модели, которые обучились с размером mini-batches равным 128. В среднем, при использовании таких моделей, получаем классификацию точнее на 2-5% по сравнению с большими размерами mini-batches.

Выводы по задаче распознавания речи

В результате экспериментов наиболее точно классифицирующие модели стекового автоэнкодера были получены при использовании стандартного подхода на обеих стадиях обучения. Однако это и наиболее затратный по времени подход. Если быстродействие системы играет важную роль, то можно заменить стандартный подход на соревновательный на обеих стадиях обучения. При этом, имея выигрыш в скорости, ошибка классификации увеличивается на 2-3%. Использование разных подходов для pre-training и fine-tuning не оптимально. При обучении модели стандартным pre-training и соревновательным fine-tuning получаем практически такой же результат, как и при применении на обеих стадиях соревновательного подхода. При этом

имеется существенный проигрыш во времени. При обучении модели соревновательным pre-training и стандартным fine-tuning получаем результат на 5-7% менее точный, чем при использовании в обоих случаях стандартного подхода и на 2-4% менее точный, чем при использовании соревновательного подхода на обеих стадиях. При этом имеется проигрыш по времени по сравнению с соревновательным подходом.

Заключение

В данной работе изучены основные понятия, связанные с нейросетями. Более детально исследованы следующие модели: логистическая регрессия, многослойный персептрон, стандартный автоэнкодер, шумоподавляющий автоэнкодер, стековый шумоподавляющий автоэнкодер, скрытые марковские модели. Для проведения экспериментов реализованы модели стекового шумоподавляющего автоэнкодера с тремя классификаторами: логистическая регрессия и два типа скрытых марковских моделей (подход “распознавание слов” и подход “распознавание речи”). Каждая из этих моделей обучалась с целью подбора оптимальных параметров для анализа потоковых данных.

В первой части работы проанализированы возможности применения скрытых марковских моделей для анализа потоковых данных. Эксперименты проводились для задачи распознавания фаз сна. Однако высокой точности достичь не удалось. Причина такого результата может заключаться в том, что функция стоимости останавливается в локальном минимуме, так и не достигнув глобального. В связи с этим встает вопрос о выборе альтернативного подхода к обучению.

Во второй части работы сравнивался стандартный подход к обучению нейросетей с соревновательным. Эксперименты проводились на модели стекового шумоподавляющего автоэнкодера с логистической регрессией в качестве классификатора. Лучшие результаты были получены при обучении стандартным pre-training и стандартным fine-tuning. Однако это и наиболее затратный по времени подход. Применение менее затратного по времени соревновательного подхода на обеих стадиях обучения ухудшает классификации на 2-3%. Использование разных подходов к обучению на стадиях pre-training и fine-tuning не дает лучших результатов по сравнению с одинаковыми подходами на обеих стадиях обучения.

Список литературы

[1] M. Borazio, E. Berlin, N. Kucukyildiz, P. Scholl and K. V. Laerhoven, “Towards Benchmarked Sleep Detection with Inertial Wrist-worn Sensing Units”, IEEE International Conference on Healthcare Informatics, IEEE Press, Sep. 2014.

[2] Lawrence R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”, Proceeding of the IEEE, Feb. 1989, pp. 257-286.

[3] N. Oakley, “Validation with Polysomnography of the Sleepwatch Sleep/Wake Scoring Algorithm used by the Actiwatch Activity Monitoring System,” Technical Report, 1997.

[4] R. J. Cole, D. Kripke F, D. Mullaney, and C. Gillin, “Automatic Sleep/wake Identification from Wrist Activity,” Sleep, vol. 15, no. 5, 1992, pp. 461–469.

[5] Y. Bengio and P. Frasconi, “An input output HMM architecture”, Advances in Neural Information Processing Systems, vol. 7, MIT Press, 1995.

[6] E. Hsu, K. Pulli, J. Popovic, “Style Translation for Human Motion”, ACM Transactions on Graphics, vol. 24 n.3, July 2005.

[7] N. Sukhorukova, A. Stranieri, B. Ofoghi, P. Vamplew, M. Saleem, L. Ma, A. Ugon, J. Ugon, N. Muecke, H. Amiel, C. Philippe, A. Bani-Mustafa, S. Huda, M. Bertoli, P. Lévy, J.-G. Ganascia, “Automatic sleep stage identification: difficulties and possible solutions”, Proceedings of the Fourth Australasian Workshop on Health Informatics and Knowledge Management, vol. 108, Jan. 2010, pp. 39-44.

[8] James Howard, William Hoff, “Forecasting Building Occupancy Using Sensor Network Data”, Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, Aug. 2013, pp. 87-94.

[9] Johannes Stigler, Fabian Ziegler, Anja Gieseke, J.Christof M.Gebhardt, Matthias Rief, “The Complex Folding Network of Single Calmodulin Molecules”, Science, vol. 334, 28 Oct. 2011, pp. 512-516.

[10] Wing Wong, Mark Stamp. “Hunting for metamorphic engines”, Journal in Computer Virology, vol. 2, Issue 3, Dec. 2006, pp. 211-229.

[11] Ka-Chun Wong, Tak-Ming Chan, Chengbin Peng, Yue Li and Zhaolei Zhang, “DNA Motif Elucidation using Belief Propagation”, Nucleic Acids Research, vol. 41, Issue 16, 2013, pp. e153.

[12] Christopher Nicolai and Frederick Sachs, “Solving ION Channel Kinetics with the QuB Software”, Biophysical Reviews and Letters, Available Issues, vol. 08, Issue 03n04, Dec. 2013.

[13] Miguel A. Carreira-Perpinan, Weiran Wang, “Fast algorithms for learning deep neural networks”, Bay Area Machine Learning Symposium, 30 August 2012.

[14] Geoffrey E. Hinton, Simon Osindero, Yee-Whye Teh, “A fast learning algorithm for deep belief nets”, Neural Computation, vol.18, Issue 7, July 2006, pp. 1527-1554.

[15] Ronan Collobert, Jason Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning”, Proceedings of the 25th International Conference on Machine Learning (ICML), 2008, pp. 160-167.

[16] James Martens, “Deep learning via Hessian-free optimization”, Proceedings of the 27th International Conference on Machine Learning (ICML), 2010.

[17] Quoc V. Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, Andrew Y. Ng, “On Optimization Methods for Deep Learning”, Proceedings of the 28th International Conference on Machine Learning (ICML), 2011.

[18] С. В. Траньков, В. А. Яворский, М. Бородовский, “Анализ эффективности использования семейства алгоритмов GeneMark при аннотации геномов”, Труды МФТИ, том 4, №2 (14), 2012, стр. 202-209.

[19] Р. Каллан, “Основные концепции нейронных сетей”, Вильямс, 2001.

[20] И.Р.Двойной, И.И.Сальников, “Сравнительный анализ структур скрытых марковских моделей, используемых в задаче установления личности человека по изображению лица”, Современные проблемы науки и образования, №6, 2013.

[21] И.В.Заенцев, “Нейронные сети: основные модели”, Учебное пособие к курсу “Нейронные сети”, 1999.

[22] Ф. Уоссермен, “Нейрокомпьютерная техника: Теория и практика”. Перевод на русский язык, Ю. А. Зуев, В. А. Точенов, 1992, гл. 1, 2, URL: http://neuroschool.narod.ru/books/nntech.html (дата обращения: 10.03.2014).

[23] Unsupervised Feature Learning and Deep Learning Tutorial, URL: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial (дата обращения: 15.03.2015).

[24] DeepLearning 0.1 Documentation, URL: http://deeplearning.net/tutorial/contents.html (дата обращения: 20.03.2015).